ZEUS-3D USER MANUAL

Version 3.2.1

David A. Clarke*, Michael L. Norman,

and Robert A. Fiedler

Laboratory for Computational Astrophysics

National Center for Supercomputing Applications

University of Illinois

Revised August 17, 1994

© 1994, The Board of Trustees of the University of Illinois

*Current address: St. Mary's University, Department of Astronomy, Halifax, NS B3HC3H Canada

TABLE OF CONTENTS

Page

I Copyright Notice . . . 3

II Introduction . . . 4

III Running ZEUS-3D . . . 8

3.1 Overview . . . 8

3.2 The Macro File `zeus32.mac` . . . 9
3.2.1 The EDITOR Definitions . . . 11
3.2.2 The EDITOR Aliases . . . 12

3.3 The Script File `zeus32.s` . . . 14
3.3.1 Files Copied from AFS . . . 16
3.3.2 Creating the `zeus3.2` Directory . . . 17
3.3.3 Creating the Change Deck `chgzeus` . . . 17
3.3.4 Preprocessing `zeus32` . . . 18
3.3.5 Creating the Input Deck `inzeus` . . . 19
3.3.6 Making the Executable `xzeus32` . . . 20
3.3.7 Cleaning up Compiler Listings . . . 20

3.4 Executing ZEUS-3D . . . 20

IV Output from ZEUS-3D . . . 22

V Interacting with ZEUS-3D . . . 28

VI Adding Source Code to ZEUS-3D . . . 30

6.1 Adding an Entire Subroutine . . . 30
6.2 Microsurgery Using EDITOR . . . 35

VII Quick Summary . . . 39

Appendix 1: ZEUS-3D Skeleton . . . 40

Appendix 2: The Namelists . . . 42

Appendix 3: The ZEUS-3D Variables . . . 71

Index . . . 77

I COPYRIGHT NOTICE

Permission to use, copy, modify, and distribute this software and its documentation for education, research, and non-profit purposes is hereby granted provided that the copyright notice and the unit identification (Laboratory for Computational Astrophysics, NCSA) along with this permission notice appear in all such copies and supporting documentation; that no charge be made for such copies; and that the name of the University of Illinois not be used in advertising or publicity pertaining to the distribution of this software without specific prior written permission. Any entity desiring permission to incorporate this software into commercial products should contact NCSA at the University of Illinois at Urbana-Champaign (217-244-0072). The University of Illinois makes no representations about the suitability of this software for any purpose. It is provided "as is" without expressed or implied warranty.

It is the intent of the Laboratory for Computational Astrophysics to make its codes available to the widest possible user community for solving and visualizing computationally complex fluid dynamics problems in a diverse arena of applications. We request that everyone who uses ZEUS-3D register with the LCA at lca@ncsa.uiuc.edu so that we may maintain a complete list of our users and a description of the problems to which they are applying our code.

II INTRODUCTION

2.1 VERSION 3.2.1

ZEUS-3D is a three-dimensional MHD code based on a two-dimensional one. ZEUS-3D was designed with astrophysical applications in mind, although problems in the other physical sciences can be addressed with this software. The code uses algorithms and structures developed over the past 20 years by David Clarke, Philip Colella, Chuck Evans, John Hawley, Michael Norman, Larry Smarr, Jim Stone, Bram van Leer, Jim Wilson, Karl-Heinz Winkler, Paul Woodward, and others. It is therefore appropriate that this code should become a community tool, available to anyone with a complex fluid dynamics problem to solve.

ZEUS-3D is a three-dimensional ideal (non-resistive, non-viscous, adiabatic) non-relativistic magnetohydrodynamical (MHD) fluid equation solver which numerically integrates the following coupled partial differential equations as a function of time and space:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = 0 \tag{1}$$

$$\frac{\partial \mathbf{S}}{\partial t} + \nabla \cdot (\mathbf{S}\mathbf{v}) = -\nabla p - \rho \nabla \Phi + \mathbf{J} \times \mathbf{B} \tag{2}$$

$$\frac{\partial e}{\partial t} + \nabla \cdot (e\mathbf{v}) = -p \nabla \cdot \mathbf{v} \tag{3}$$

$$\frac{\partial \mathbf{B}}{\partial t} = \nabla \times (\mathbf{v} \times \mathbf{B}) \tag{4}$$

where:

ρ = matter density
$\mathbf{v}$ = velocity flow field
$\mathbf{S}$ = $\rho\mathbf{v}$ = momentum vector field
p = thermal pressure
Φ = gravitational potential
$\mathbf{J}$ = current density
$\mathbf{B}$ = magnetic induction
e = internal energy density (per unit volume)

This code is strictly Newtonian. Relativistic astrophysics cannot be simulated in any way with this version. No explicit account for relativistic particles is incorporated either. The code assumes strict charge neutrality at all times – it is not a plasma code. It is assumed that the fluid is thermal, and is coupled to the magnetic field via collisions with an ionized component which never undergoes charge separation. The thermal pressure is assumed to be isotropic.

It is assumed that the user is familiar with the fundamentals of MHD and has come up with a complex problem to solve which is described by equations 1 through 4. It is

also assumed that the user has a working knowledge of some flavor of UNIX, such as Cray UNICOS, SGI Irix, etc. In this spirit, this manual is designed to instruct the user on the mechanics of using ZEUS-3D.

ZEUS-3D has the following features:

- finite differencing on an Eulerian mesh (possibly moving in an average sense with the fluid);
- fully explicit in time and therefore subject to the Courant condition;
- operator and directional splitting of the equations governing the hydrodynamic variables;
- can be used efficiently for 1-D and 2-D simulations with any of the coordinates reduced to symmetry axes;
- Cartesian, cylindrical, and spherical coordinates for 1-D, 2-D, and 3-D simulations;
- written in a "covariant" fashion to minimize the effects of the different coordinate systems on the structure of the code;
- fully staggered grid, with scalars (density and internal energy) zone-centerd and vector components (velocity and magnetic field) face-centered (Derived vector components (current density and emf's) are edge-centered.);
- von-Neumann Richtmyer artificial viscosity to smear shocks;
- upstream-weighted, monotonic interpolation using one of donor cell (first order), van Leer (second order), or piecewise parabolic advection (third order) schemes;
- "Consistent Advection" (Michael Norman) used to evolve internal energy and momenta; and
- "Constrainted Transport" (Evans and Hawley) modified with the Method of Characteristics (Jim Stone) is used to evolve the magnetic fields.

NEW FEATURES FOR VERSION 3.2.1 (COMPARED TO 3.2)

- A much more powerful zeus32.s script makes installing and customizing ZEUS-3D much easier.
- ZEUS-3D and EDITOR have been modified to work properly on Cray, Convex, Sun, and SGI systems all over the world. FORTRAN source code for the standard library routines `ISMIN`, `ISMAX`, and `SASUM` are provided in zeus32.s for systems without them.
- The EDITOR aliases `UY` and `U2` have been replaced by the `CALMATH` alias, which should be defined only for UNICOS systems that have the Cray "calmath" math library.

- Line-of-sight integrations through the data volume for a variety of variables, including the Stokes parameters (see Section IV) may be output in both `XYZ` and `ZRP` coordinates. (The EDITOR definition `RADIO` must be set to invoke this display option.)
- An option for generating "time slice" data, that is, values of certain global quantities as a function of time. (The EDITOR definition `TIMESL` must be set in order to get time slice output.)
- Rather than generating polar pixel dumps, ZEUS-3D converts polar slices to Cartesian slices "on the fly" before writing them to disk.
- 1-D and 2-D NCAR graphical output.
- An EDITOR alias `FINISH` which represents a subroutine called after the main loop of the main program `zeus3d` gives the user a slot in which to perform various tasks at the end of the run.
- The code can be microtasked automatically for Crays. Tests indicate that for typical runs, a real-time speed-up of 3.9 can be achieved with 4 dedicated processors.
- The code will now run efficiently (*i.e.*, it vectorizes) as a unitasked process on Convex C Series machines. This is done by defining the EDITOR definition `CONVEXOS`. Multitasking on a Convex using the `-O3` option can be done, but yields a real-time speed-up of only about 2.5 on a four processor machine.
- More combinations of dimension and geometry have now been tested against analytical solutions. The list includes Cartesian (`XYZ`) with two, one, or no symmetry flag(s) set, cylindrical (`ZRP`) with either `JSYM+KSYM` or `KSYM` set, spherical polar (`RTP`) with either `JSYM+KSYM` or `KSYM` set. Other combinations should be used with caution.
- One can select an isothermal equation of state. An EDITOR definition `ISO` causes the code to take advantage of the reduction in memory and computation required for isothermal systems.
- Yu Zhang (NCSA) has implemented a 3-D self-gravity module using the so-called DADI (Dynamical Alternating Direction Implicit) scheme. The EDITOR definition `GRAV` must be set if self-gravity is to be invoked.
- One has the choice of solving either the total energy equation (to globally conserve the total energy) or the internal energy equation. The toggle `itote` in the namelist `hycon` specifies which of these formalisms is to be used (Byung-Il Jun, NCSA).
- Pixel, Voxel and RADIO dumps may be made in HDF format. This avoids the cumbersome process of "bracketing" the images, but at the cost of more than four times the disc space requirements.
- It is possible to read a restart dump, for example, that was generated by a compiled version of the code with different EDITOR macro settings and different values for the array parameters.

- Boundaries must be regular.

Users of version 3.0 will be happy to note that there are no major changes in the way ZEUS-3D is compiled or executed, and the namelist parameters have remained more or less fixed. Still, there are enough subtle changes that it might do the experienced user some good to review these notes before attempting to run a job with this new version. Also note that version 3.2 cannot read restart dumps created by version 3.0, and vice versa.

III RUNNING ZEUS-3D

3.1 OVERVIEW

ZEUS-3D version 3.2.1 runs on most Cray, Convex, SGI, and Sun systems. The examples in this manual are written assuming that the user will run the code on a Cray, while providing some additional notes for running the code on other systems. The user may obtain assistance with porting the code by contacting the Laboratory for Computational Astrophysics (LCA) at lca@ncsa.uiuc.edu.

In order to obtain the ZEUS-3D executable, download a script called zeus32.s from landrew.ncsa.uiuc.edu. Establish an ftp connection (not telnet) using the signon `zeus3d` and the current password (available from the LCA to registered users), and cd to the directory:

```
/afs/ncsa/projects/lca/codes/zeus3d/source
```

The on-line version of zeus32.s is much more eloborate than the one shown below for illustrative purposes. Running this script will automatically download everything else needed to create the EDITOR executable, build the `namelist` and other libraries, and perform the test problem discussed here. To customize ZEUS-3D for applications of your own, you need to modify the EDITOR "changedeck" `chgzeus`, which consists of the macro definition file `zeus32.mac`, instructions to set various array sizes, and any supplimental source code modules. Rerunning the script will then produce the corresponding ZEUS-3D executable. The next two subsections describe these modifications in more detail.

The `zeus32.s` script is run by typing:

```
chmod 700 zeus32.s (To make sure it is executable.)
zeus32.s
```

This script performs sequentially the following tasks:

- retrieves all the files necessary to compile the code;
- creates a directory called `zeus3.2` within the user's current directory to store all the source and object files created during compilation;
- creates a change deck for the ZEUS-3D code containing preprocessor macros and aliases (`zeus32.mac`, next subsection), and changes to the source code (if any) required for the application (The most common and often the only changes which must be made to the source code are to the parameter statements which set the size of the arrays needed for the run.);
- runs the EDITOR preprocessor;
- creates the input deck for the ZEUS-3D run; and finally
- compiles and loads the source code and libraries (using the UNIX "make" facility).

A description of the file naming convention may be helpful at this point. ZEUS-3D refers in a general way to the package and its capabilities while `zeus32` is more specific, and is a mnemonic for "ZEUS-3D, version 3.2". `zeus32` is the common denominator for the names of the principle files required to create the executable. Thus, the source code itself is simply `zeus32`, the script file is `zeus32.s`, the macro file is `zeus32.mac`, and the executable is `xzeus32`. However, some minor files don't follow this convention. The input deck is just `inzeus` – no "32" suffix. There are two change decks – one is `chgzeus`, the other is `chgz32` and the libraries don't even have ZEUS as part of their names. The bottom line is that if the only changes to be made to the source code are the values of the parameters which set the array dimensions, then there are only three files to be concerned with: `chgzeus`, `zeus32.mac`, and `inzeus`. The rest is automatic.

3.2 THE MACRO FILE `zeus32.mac`

Below is an example of a `zeus32.mac` file for use on a Cray Y-MP with the calmath library. This file is much less general than the on-line version, but it still cantains all the macros and runs the same test problem. Another example provided on-line is the 3-D blast-wave test problem set up by the version of `zeus32.s` called `blast.s`, which can be downloaded from landrew.ncsa.uiuc.edu as described earlier from directory:

`/afs/ncsa/projects/lca/codes/zeus3d/tests/3d/blast`

The on-line versions of zeus32.s are designed to create the ZEUS-3D executable on various systems besides Cray UNICOS.

```
**===*====1====*====2====*====3====*==*====3====*====2====*====1====*====
**                                                                      **
*****************  CONDITIONAL COMPILATION SWITCHES  *****************
**                                                                      **
**  1) symmetry axes:  ISYM, JSYM, KSYM
**
*define   KSYM, JSYM
**
**  2) geometry:  XYZ, or ZRP, or RTP
**
*define   XYZ
**
**  3) physics:  MHD, ISO, GRAV
**
*define   MHD
**
**  4) algorithms:  MOC
**
*define   MOC
**
**  5) data output modes:  PLT1D, PLT2D, PIX, VOX, HDF, DISP, RADIO,
**                         TIMESL
**
```

```
*define   PLT1D
**
**  6) operating system:  UNICOS, CALMATH, CONVEXOS, SUNOS, IRIX
**
*define   UNICOS, CALMATH
**                                                                    **
************************  MODULE NAME ALIASES  ************************
**                                                                    **
**  The modules "BNDYUPDATE", "SPECIAL", "SPECIALSRC", "SPECIALTRN",
**  "USERDUMP", and "FINISH" are slots available to the user to help
**  adapt the code to the problem being solved.
**
*alias    START            mstart
*alias    BNDYUPDATE       empty
*alias    EXTENDGRID       empty
*alias    GRAVITY          empty
*alias    SPECIAL          empty
*alias    SOURCE           srcstep
*alias    SPECIALSRC       empty
*alias    TRANSPORT        transprt
*alias    SPECIALTRN       empty
*alias    NEWTIMESTEP      nudt
*alias    NEWGRID          empty
*alias    DATAOUTPUT       dataio
*alias    FINISH           empty
**
*alias    PROBLEM          shkset
*alias    ATMOSPHERE       empty
*alias    PROBLEMRESTART   empty
*alias    ARTIFICIALVISC   viscous
*alias    USERDUMP         empty
**                                                                    **
************************  ERROR CRITERIA ALIASES  ***********************
**                                                                    **
*alias    GRAVITYERROR     1.0e-6
*alias    GRIDERROR        1.0e-6
*alias    PDVCOOLERROR     1.0e-6
*alias    NEWVGERROR       1.0e-10
**                                                                    **
************************  ITERATION LIMITS ALIASES  *********************
**                                                                    **
*alias    GRAVITYITER      5
*alias    GRIDITER         20
*alias    PDVCOOLITER      20
*alias    NEWVGITER        20
```

These are all preprocessor commands (the preprocessor used here is called EDITOR, and was developed by David Clarke), and become part of the "change deck" `chgzeus` created by the script file `zeus32.s`. A change deck is a file which is merged with the source code during the preprocessing step of `zeus32.s`. Both the source code and the change deck can contain preprocessor commands which are interpreted, carried out, and

then deleted from the code by EDITOR before the code is compiled by the FORTRAN compiler. All preprocessor commands have an asterisk (*) in column 1. Double asterisks indicate a comment. When the preprocessor has finished, the result is a purely FORTRAN source code tailored specifically for the problem to be solved. Therefore, in order to customize the code for the problem at hand, it is necessary to set the EDITOR "definitions" and "aliases" (generically referred to as "macros") found in `zeus32.mac`.

The combined effect of the macros is two-fold. First, they determine what parts of the code are activated and what parts are ignored. Thus, it is possible to eliminate the computations and the memory requirements necessary to evolve the magnetic fields, for example, by not defining the `MHD` macro [this can be done by "commenting out" (double asterisk) the `*define MHD` statement in the example above]. The preprocessor will then remove all coding pertaining to magnetic fields including the declarations of the magnetic field arrays during the preprocessing step. The compiler never sees the magnetic stuff, and the executable is streamlined for the hydrodynamics problem. Of course, the original source code is not altered by preprocessing it. Rather, the preprocessor creates a precompiled version of the code and stores each subroutine into its own file [to facilitate debugging and "make"] in the directory `zeus3.2` which was created by the script file `zeus32.s`. Second, the alias macros can be used to substitute any character string in the code during the preprocessing step.

3.2.1 The EDITOR Definitions

A description of the definition macros (called "Conditional Compilation Switches" in the example) follows:

1. The code can be streamlined (optimized) for 1-D and 2-D problems by setting the appropriate symmetry macros. If symmetry along any of the i (x1), j (x2), or k (x3) axes is desired, then set the `ISYM`, `JSYM`, or `KSYM` macros. If the macros are not set and a 1-D or 2-D calculation is initialized by the input deck, ZEUS-3D will still carry out the sub 3-D computation correctly, but will do so less efficiently.

2. The geometry is set by setting ONE of the `XYZ` (Cartesian), `ZRP` (cylindrical), or `RTP` (spherical polar) macros. These macros are mutually exclusive, so set only one of them!

3. By setting the `MHD` macro, the algorithm for evolving the magnetic fields is activated. With `MHD` on, additional field arrays are declared and the code pertaining to updating the magnetic field is compiled. The `ISO` macro should be set if an isothermal equation of state is desired. With `ISO` defined, the internal energy variables are not declared saving both computational time and memory. Finally, defining `GRAV` *and* setting the EDITOR alias `GRAVITY` to `gravity` (see below) will turn on the DADI self-gravity module developed by Yu Zhang at NCSA.

4. The MoC algorithm (Method of Characteristics, developed by Jim Stone and Michael Norman of NCSA, John Hawley at the University of Virginia, and Chuck Evans at the University of North Carolina) is the most robust algorithm developed for evolving magnetic fields, and is essential for stable propagation of Alfvén waves. Thus, it is

advisable that the macro `MOC` be set for most applications. In some circumstances, especially for dynamically unimportant (passive) magnetic fields, stable Alfvén waves are not critical, and `MOC` may be turned off yielding a time savings of 25% to 30%. The user should be warned, however, that without MoC, numerical instabilities in passive magnetic fields have been observed to generate local enhancements of many orders of magnitude yielding dynamically important magnetic fields in a relatively short period of time.

5. The graphics enabled during a run are set by the graphics macros. To enable 2-D pixel dumps, set `PIX`, for 3-D voxel dumps, set `VOX`, for 1-D NCAR plots, set `PLT1D`, for 2-D NCAR plots, set `PLT2D`, for HDF dump files, set `HDF`, for display dumps, set `DISP`, for RADIO dump files, set `RADIO`, and for time slice dumps, set `TIMESL`. As many as these may be set simultaneously as necessary. See Section IV for a discussion of the various ZEUS-3D dump files.

6. The operating system is defined by setting only one of the macros `UNICOS` (for the Crays) or `CONVEXOS` (for the Convex), etc. In addition, `CALMATH` should be set for Crays that have the calmath library.

3.2.2 The EDITOR Aliases

The alias macros allow phrases in the code to be substituted for other phrases during the precompiling step. Thus, "Module Name Aliases" give the user control over what subroutines are called during execution. As an example, in the main program of the source code, there is a statement: `call START` which becomes `call mstart` after preprocessing using the above example of the file `zeus32.mac`. Note that there is no subroutine called `START` but there is a subroutine in the source code called `mstart`. Thus, the user is free, in principle, to create his or her own initialization subroutine to be called instead of `mstart` which can be linked into the code by altering the alias setting for `START` from `mstart` to the name of the user's initialising subroutine. Note that by setting any of the Module Name Aliases to `empty` (a subroutine in `zeus32` which does nothing but return to the calling routine), a Module Name Alias can be effectively "turned off".

Aliases can also be used to set parameters in various parameter statements scattered throughout the source code. These are the "Error Criteria Aliases" and "Iteration Limits Aliases" in the example above. Thus the EDITOR statement:

```
alias GRAVITYERROR     1.0e-5
```

sets the maximum convergence error in the self-gravity module to 10^{-5}. Somewhere in the code is the statement `parameter ( maxit = GRAVITYITER )` and the preprocessor makes the substitution. However, the majority of the parameters (array dimensions, for example) are set directly in `zeus32.s` which is described in the next subsection.

To better understand the descriptions of the "Module Name Aliases" which follow, the reader should examine the flow chart in Appendix 1 (the "ZEUS-3D Skeleton"). This is a flow chart of the code, and indicates in which order the Module Name Aliases are called. Some subroutines are charged with reading the input data from the input deck `inzeus`. A description of all the input parameters is given in Appendix 2.

1. `START`: This module is called just once before the computations begin. It should initialize all the variables to be used in the simulation and perform all the initial I/O. Currently, the only choice available for `START` is `mstart`.

2. `BNDYUPDATE`: This module is called at the beginning of each time step and allows inflow boundary conditions to be evolved in time should this be necessary for the simulation. Examples of evolving inflow boundary conditions include helically perturbing the inflow at a jet orifice to break the symmetry (`wiggle`), generating magnetic field at the boundary (`bgen`), or `empty` if no inflow boundary update is desired. The user can, of course, supply a subroutine for this alias. See Section VI for discussion on how to add a subroutine to the code.

3. `EXTENDGRID`: This module will allow the grid to be extended as a disturbance (shock) propagates into initially quiescent zones. Currently, the only options are `extend` and `empty`. The subroutine `extend` will prevent quiescent zones from being updated until the disturbance comes within five zones, potentially saving significant amounts of computational time. Care should be exercised in its use, however. If the subroutine is unsuccessful in determining when the disturbance gets close to an edge of the current computational domain, the results can be disastrous.

4. `GRAVITY`: This module updates the self-gravitational potential. Currently, the only choices are `empty` and `gravity`. The latter invokes the DADI algorithm developed for ZEUS-3D by Yu Zhang, and should still be considered experimental. In most applications, the DADI algorithm will consume a similar amount of cpu time as the hydrodynamics.

5. `SPECIAL`: This is a simplistic solution to the potentially complex problem of the user desiring to add a whole new type of physics to the code. It assumes that changes do not need to be intertwined into existing modules, which in practice, often will be necessary. The two accompanying "plugs" `SPECIALSRC` (for "special" source terms) and `SPECIALTRN` (for "special" transport considerations) allow for some flexibility in installing new physics within the current structure, but this still may not be enough for any type of sophisticated addition. By default, all three are set to `empty`.

6. `SOURCE`: This is the module in which source terms are incorporated. For full dynamics, this should be set to `srcstep` (or the user's module if need be) while for problems of pure advection, this should be set to `empty`.

7. `SPECIALSRC`: See `SPECIAL`.

8. `TRANSPORT`: This is the module for the transport of variables across zone boundaries and should be set to `transprt` or to the user's equivalent module. It is unlikely that `empty` should ever be used here.

9. `SPECIALTRN`: See `SPECIAL`.

10. `NEWTIMESTEP`: This module determines how the next time step is computed. Since ZEUS-3D is an explicit code, all algorithms should incorporate the Courant condition (CFL). Current choices are `nudt` for full (M)HD problems, and `advectdt` for pure advection problems.

11. NEWGRID: This is the module which adjusts the grid should grid velocities be desired to partially follow the flow. Current choices are `newgrid` and `empty`. In practice, users will probably have to provide their own prescriptions for evaluating the grid velocities, as most of the available methods are untested. This will require replacing or adding to the subroutine `newvg`. See Section VI for discussion on how to add or modify a subroutine in the code.

12. DATAOUTPUT: This module is responsible for data I/O. Setting this macro to `dataio` will cause restart dumps, NCAR plot files, pixel dumps, voxel dumps, hdf files, display files, RADIO dumps, time slice dumps, and any other format as specified by the macro `USERDUMP` to be created at time intervals set by the user (Section IV). Setting the macro to `empty` will prevent all data I/O – probably not a good idea.

13. FINISH: This is a "plug" available to the user to have any user-supplied subroutine called once at the end of execution. It could, for example, be used to generate NCAR plots of certain variables that the user has been monitoring via another user-supplied subroutine to which `USERDUMP` has been set.

14. PROBLEM: This macro specifically initializes all the flow variables and is called by the subroutine `setup`, which is called by `mstart` (`START`). It will almost always be set to a user-written subroutine, unless, by some miracle, the user can use a problem generator which already exists in the source code. The problem generators currently available in `zeus32` include `jetinit`, which initializes the variables for the launching of a jet, `blast`, which initializes the variables for an explosion, `advect`, which initializes the variables for 1-D advection tests, and `shkset`, which initializes the variables for 1-D Sod shock tube tests.

15. ATMOSPHERE: This macro defines the atmosphere for a jet, and is called by `jetinit`. For a uniform atmosphere, set `ATMOSPHERE` to `empty`, since a uniform atmosphere is established in `jetinit` before `ATMOSPHERE` is called. Otherwise, the user will have to supply a subroutine to initialize the desired atmosphere.

16. PROBLEMRESTART: This macro allows the specifications of the problem to be altered should the job be restarted from a restart dump. Set the macro to `empty` if no alteration of the problem is desired (as, for example, to simply extend the evolution time).

17. ARTIFICIALVISC: This macro specifies which artificial viscosity algorithm should be used. Currently the only option is `viscous`, which uses either the von-Neumann Richtmyer artificial viscosity algorithm or a form which is linear in velocity.

18. USERDUMP: See `DATAOUTPUT`.

3.3 THE SCRIPT FILE `zeus32.s`

Below is a stripped-down version of the `zeus32.s` script. Download the much more eloborate version from landrew.ncsa.uiuc.edu via ftp as described earlier.

It is assumed below that the user's Cray is on AFS (Andrew File System) and that the command "`klog zeus3d -cell ncsa.uiuc.edu`" has been issued to allow access to files on landrew. The script file is run by typing:

```
chmod 700 zeus32.s
zeus32.s.
```

```
#!/bin/csh -v
#============ SCRIPT FILE TO CREATE THE ZEUS-3D EXECUTABLE ============#
#
#=================================================> Get files from AFS.
set ZEUS3D = "/afs/ncsa/projects/lca/codes/zeus3d"
if (! -e zeus32) cp $ZEUS3D/source/zeus32 .
if (! -e zeus32.mac) cp $ZEUS3D/xeq/uy/zeus32.mac .
#if (! -e chgz32) cp $ZEUS3D/xeq/uy/chgz32 .
if (! -e xedit21) cp $ZEUS3D/editor/xeq/uy/xedit21 .
if (! -e namelist.lib) cp $ZEUS3D/namelist/xeq/uy/namelist.lib .
if (! -e cfiles.lib) cp $ZEUS3D/cfiles/xeq/uy/cfiles.lib .
if (! -e ncar.libo) cp $ZEUS3D/ncar/xeq/uy/ncar.libo .
#========================> If necessary, create the directory "zeus3.2".
if (! -e zeus3.2) mkdir zeus3.2
#-------------------------------------------------> Create the change deck.
\rm -f chgzeus
cat << EOF > chgzeus
*read zeus32.mac
*d par.14,15
       parameter     ( in   = 805, jn   =   1, kn   =   1, ijkn = 805 )
       parameter     ( nxpx =   1, nypx =   1, nxrd =   1, nyrd =   1 )
**read chgz32
EOF
#======================> Create the input deck for EDITOR, and execute.
\rm -f inedit
cat << EOF > inedit
 \$editpar   inname='zeus32'
          , ibanner=0, idump=1, job=3, safety=0.20
          , ipre=1, inmlst=1, iupdate=1, iutask=0
          , chgdk='chgzeus'
          , branch='zeus3.2'
          , makename='makezeus', xeq='xzeus32', loader='ncargf77'
          , compiler='cft77 -M0726'
c         , coptions='-ez -o off'
c         , compiler='cf77'
c         , coptions='-c -Zp -Wf"-M0726"'
          , libs='-lcalmath namelist.lib cfiles.lib ncar.libo -ldf $
EOF
xedit21
#---------------------------------> Create the input deck for ZEUS-3D.
\rm -f inzeus
cat << EOF > inzeus
 \$iocon    iotty=6, iolog=2                                              \$
 \$rescon   dtdmp=80.0, id='xd', resfile='zr00xd'                         \$
 \$ggen1    nbl=800, x1min= 0.0, x1max=800.0, igrid= 1, x1rat=1.0
```

```
        , lgrid=.t.                                                    \$
 \$ggen2    nbl=001, x2min= 0.0, x2max=  1.0, igrid= 1, x2rat=1.0
        , lgrid=.t.                                                    \$
 \$ggen3    nbl=001, x3min= 0.0, x3max=  1.0, igrid= 1, x3rat=1.0
        , lgrid=.t.                                                    \$
 \$pcon     nlim= 999999, tlim=80.0, ttotal=  900.0, tsave=10.0         \$
 \$hycon    qcon=2.0, qlin=0.0, courno=0.5, iord=2, istp=0, itote=0     \$
 \$iib      niib(1:  1,1:  1)=2                                         \$
 \$oib      noib(1:  1,1:  1)=2                                         \$
 \$ijb      nijb(1:  1,1:805)=2                                         \$
 \$ojb      nojb(1:  1,1:805)=2                                         \$
 \$ikb      nikb(1:805,1:  1)=2                                         \$
 \$okb      nokb(1:805,1:  1)=2                                         \$
 \$grvcon                                                               \$
 \$eqos     gamma=2.0                                                   \$
 \$gcon                                                                 \$
 \$extcon                                                               \$
 \$plt1con  iplt1dir=1, dtplt1=80.0
        , plt1var= 'd ', 'se', 'p ', 'v1', 'v2', 'v3', 'b1', 'b2'
        , 'b3', 'bd'                                                   \$
 \$plt2con                                                              \$
 \$pixcon                                                               \$
 \$voxcon                                                               \$
 \$usrcon                                                               \$
 \$hdfcon                                                               \$
 \$tslcon                                                               \$
 \$discon                                                               \$
 \$radcon                                                               \$
 \$pgen     idirect=1, n0=400, d0=1.000, e0=1.0, v10=0.0, b10=0.75
        , b20=0.6, b30=0.8                                             \$
 \$pgen     idirect=1, n0=400, d0=0.125, e0=0.8, v10=0.0, b10=0.75
        , b20=-0.6, b30=-0.8                                           \$
EOF
#=============================================> make the ZEUS executable.
make -f makezeus
```

Note that a # in column 1 indicates a comment in a script file. In this example, two flavors of comment lines are used. Comments led with a double dashed line (=======>) indicate portions of the script file which rarely, if ever, need to be changed by the user. Comments with a single dashed line (-------->) indicate portions of the script file that will probably need to be changed with every simulation. Below are descriptions of the seven segments found in the script file `zeus32.s`.

3.3.1 Files Copied from AFS

The first segment copies the files necessary to create the ZEUS-3D executable from AFS to the user's local directory only if they do not already exist there. In the above example, the job is to be run on NCSA's Cray Y-MP "uy". This script assumes the user has already conncted to AFS as decribed above (note: in reality, uy cannot connect to AFS; ftp is used instead). If the job is to be run on another system, the compiler and loader names and options that are written to the file `inedit` would have to be altered

appropriately.

The files retrieved are:

`zeus32`	the 45,000 lines of source code divided up into $\sim$ 160 subroutines
`zeus32.mac`	file containing all the EDITOR macros (Section 3.2)
`chgz32`	the change deck containing changes to the source code that the user deems necessary for the simulation (Segment 3.3.3). This line is commented out because we are not making any changes here
`xedit21`	the preprocessor executable
`namelist.lib`	the library of subroutines which emulate the namelist feature (Segment 3.3.5)
`cfiles.lib`	the library of C subroutines
`ncar.libo`	the library of subroutines containing calls to NCAR graphics routines

3.3.2 Creating the `zeus3.2` *directory*

The second segment creates the directory `zeus3.2` on condition that it does not already exist. The precompiled source files (one subroutine per file), the compiled object files, and all the compiler listing files will be put here.

3.3.3 Creating the Change Deck `chgzeus`

The third segment creates the change deck `chgzeus` which is merged with the source code `zeus32` during the preprocessing step. The first line in `chgzeus` reads the EDITOR macros in `zeus32.mac` using the EDITOR command `*read`. This command replaces the statement with the contents of the named file. Thus, the macros in `zeus32.mac` become part of the change deck `chgzeus`, and get merged with the source code. Next, the EDITOR command `*delete` (or `*d` for short) is used to replace lines 14 and 15 in the common deck `par` in the main source code `zeus32` with parameter statements which set the parameters to the desired values for the simulation. This is where the user should indicate the size of the arrays required for the simulation to be performed. The parameters set in the given example of the script file `zeus32.s` are:

`in`	number of zones plus 5 in the 1-direction (the extra five are ghost zones)
`jn`	number of zones plus 5 in the 2-direction
`kn`	number of zones plus 5 in the 3-direction
`ijkn`	the maximum of `in`, `jn`, and `kn`
`nxpx`	maximum number of pixels in the x-direction for pixel dumps
`nypx`	maximum number of pixels in the y-direction for pixel dumps
`nxrd`	maximum number of pixels in the x-direction for RADIO dumps
`nyrd`	maximum number of pixels in the y-direction for RADIO dumps

The other parameters are discussed in Appendix 3 (Section A3.6)

Finally, the second `*read` statement (commented out in this example) inserts the file `chgz32`, which contains other changes to the source code deemed necessary by the

user to perform the computation. These changes should be specified using the language of EDITOR (very similar to HISTORIAN for those who are familiar with the old preprocessor used under CTSS), and would include additional subroutines such as the problem generator which need to be compiled with the rest of the source code. Full description of how to do this is found in Section VI.

In principle, one could manually replace the `*read zeus32.mac` command with the contents of `zeus32.mac` and replace the `*read chgz32` command with the contents of that file. Then `zeus32.s` would be the only file that would ever have to be altered. However, in the interest of modularity, the script file `zeus32.s` is presented here with the change deck divided up into three parts. The macros are all delegated to the file `zeus32.mac`, the changes to the parameters statements remain in the `zeus32.s` file where they are the most accessible, and the remaining changes to the source code are delegated to the file `chgz32`.

3.3.4 Preprocessing `zeus32`

The next segment creates the input deck for the preprocessor EDITOR and then executes it. Changes to this segment should be needed rarely. If it becomes necessary to change the name of the main source file from `zeus32`, or to change the name of the change deck from `chgzeus`, or to change the name of the directory created for the precompiled and compiled subroutine files from `zeus3.2`, or to change the name of the makefile from `makezeus`, or to change the name of the ZEUS-3D executable from `xzeus32`, or to use a compiler other than `cft77`, or to use a loader other than `ncargf77`, these changes should be reflected in the EDITOR input deck `inedit`. In addition, various compiler options can be set as necessary. For example, the compiler options (`-ez -o off`) generate a symbol table required by the debugger and turns all optimization off. These options are necessary for full debugging capabilities and can be invoked simply by "de-commenting" (deleting the "`c`" in column 1) the statement:

```
c         , coptions='-ez -o off'.
```

Note that lines "commented out" in a namelist will be echoed on the CRT as the input deck is read. This is a feature of the EDITOR namelist. (See Segment 3.3.5 and Appendix 2 for a discussion of the EDITOR namelist feature.)

If the code is to be microtasked, set `iutask` (third line of the namelist `editpar`) to 1 and invoke cf77 (rather than cft77) with the compiler options `-c -Zp -Wf"-M0726"`. This is done by "commenting out" (putting a "`c`" in column 1) the statement:

```
          , compiler='cft77 -M0726'
```

and "de-commenting" the statements:

```
c         , compiler='cf77'
c         , coptions='-c -Zp -Wf"-M0726"'.
```

Finally, additional libraries may be linked to the ZEUS-3D executable by adding them to the list `libs`.

With this input deck, the preprocessor will merge the change deck `chgzeus` with `zeus32`, carry out the precompiler commands according to the aliases and definitions

in the macro file `zeus32.mac`, split up the precompiled source code (now containing nothing but FORTRAN syntax) into separate files for each subroutine, search the directory `zeus3.2` and write to disc only those files which do not already exist or have been changed, and finally create the makefile `makezeus`, described in Segment 3.3.6.

3.3.5 Creating the Input Deck `inzeus`

The fifth segment is where the input deck for the ZEUS-3D executable is created (`inzeus`) and so the user should set all input parameters here. In this example, `inzeus` is set up for a 1-D MHD Sod shock tube problem. Note that the aliases in the example of `zeus32.mac` (previous subsection) are also set for this problem. ZEUS-3D uses namelists to specify input parameters but does not use the standard namelist utility. Historically, the first versions of namelist available under UNICOS were horrid (character variables could not be set, vectors could only be set one element at a time, error messages were unreadable), and so a more useful namelist utility was incorporated into the preprocessor EDITOR. Thus, as one of its duties, EDITOR can be instructed to replace all references to namelists with calls to subroutines found in the library `namelist.lib` which is linked to the executable during the "make" process. This step is entirely transparent to the user. Namelists can be used as always, with the usual (more or less) syntax, bearing in mind that once defined, a namelist must be read before the next namelist is defined. Since this time, the namelist utility under Cray UNICOS has been significantly improved, and should the user prefer to use the UNICOS namelist, then the input parameter `inmlst` in the EDITOR input deck `inedit` should be set to 0 (Segment 3.3.4).

One primary difference between the UNICOS namelist and the EDITOR namelist is the latter allows for rank 2 arrays to be specified in an extremely intuitive fashion. So, for example, to set `((diib1(i,j),i=20,30),j=70,80)` to 1.0, while setting the rest of the 100 by 100 array to 0.1, one merely needs to type:

```
diib1(1:100,1:100)=0.1, diib1(20:30,70:80)=1.0
```

This capacity is not supported by the current UNICOS (or CONVEXOS) version of namelist, and so some of the namelist syntax will have to be changed in the input decks `inzeus` and `inedit` should the user wish to use the standard namelist. In addition, error messages from the EDITOR namelist are much more user-friendly than from the UNICOS version. The most severe drawback of the EDITOR namelist is that it is not supported by Cray nor anyone else. In general, do not allow any of the namelist lines to extend beyond the 72nd column. The first column in each line can be a blank or a 'c' (to comment out the line) and nothing else. The second column may contain a blank or a '$' and nothing else. (Note that because `zeus32.s` is a script file, the $ must be "protected" by a \. Otherwise, the script file will try to interpret the $ as a control character rather than treating it as a character to be written to a disc file. The user will note that a \ does not precede the $ in the input deck `inzeus` once it is written to disc by `zeus32.s`.) Text specifying the input parameters may start in column 3. if a character string is too long to fit in the 72 column format (*e.g.*, `libs` in the input deck `inedit`, subsection 3.3.4), one simply types as much as one can in the first line (*i.e.*, up to the 72nd column), then

resumes typing the character string on the next line, beginning in column 3. Note the location of the single quotes.

A detailed description of all the namelist parameters is contained in Appendix 2.

3.3.6 Making the Executable `xzeus32`

The sixth segment executes the makefile `makezeus` created by the preprocessor EDITOR. The makefile will compile only those FORTRAN files in the directory `zeus3.2` which have been written since the last time they were compiled, then link all the object files together with the specified libraries (using the loader ncargf77, actually a locally available script) to create the executable `xzeus32`.

3.4 EXECUTING ZEUS-3D

Once the script file has successfully completed, simply type `xzeus32` followed by a carriage return, and ZEUS-3D will begin running. Alternatively, the executable can be run in batch mode using the NQS system. A sample NQS file called `zeus32.nqs` follows:

```
#QSUB -lT 300
#QSUB -lM 1Mw
#QSUB -lF 200Mb
#QSUB
#-------------------------------------------------> enable job accounting
ja
date
#-----> copy executable and input deck from $SAVE to batch scratch $SCR
cd $SCR
cp $SAVE/xzeus3d $SAVE/inzeus .
#-----------------------------------------------------> execute xzeus32
xzeus32
#--------------------------------------> create libraries for data dumps
tar -cvf plt2d.tar zq*
tar -cvf pixel.tar zi*
tar -cvf voxel.tar zv*
#-------------------------------------------> copy data dumps to $SAVE
cp plt2d.tar pixel.tar voxel.tar $SAVE
#--------------> report job accounting information and disable accounting
date
echo job complete
ja -csft
```

The `QSUB` directives request a maximum of 300 CPU seconds for the entire job, 1 Megaword of memory, and 200 Mbytes of temporary scratch disc space in $SCR. In this example, it was assumed that the environment variables `SAVE` and `SCR` were set to appropriate text strings in the user's `.login` file. Idealy, files that are to be saved should be transferred to a permanent storage facility.

To submit this script file to the NQS batch system, simply type:

`qsub zeus32.nqs`.

IV OUTPUT FROM ZEUS-3D

A variety of methods for dumping data to disc during execution are available in ZEUS-3D. Each of these methods has its specific use, and often all types are used simultaneously. In this section, a brief description of each method is given, along with a list of the most vital statistics. These include: the EDITOR definition (if any) which enables the data dump, the logical unit to which the dumps are attached during execution, the namelist which controls the data dump (Appendix 2), the convention used for naming the disc file for this type of data dump, and the format of the data in the disc file created.

1. RESTART DUMPS – These are full precision dumps of all variables at specified time intervals which can be used to resume a calculation should a job terminate prematurely for whatever reason. Execution can be instructed to overwrite the previous even (odd) numbered dump with the new even (odd) numbered dump should disc space be at a premium. Thus, only two restart dumps would exist at any one time. Anticipate that the size of the restart dumps will be about $10 \times$ `in` $\times$ `jn` $\times$ `kn` words for MHD runs and $6.5 \times$ `in` $\times$ `jn` $\times$ `kn` words for HD runs.

The first data written to a restart dump are the array dimensions and parameters which indicate which EDITOR macros are defined. Values of EDITOR aliases are not stored. These, then, are the first data read from a restart dump and are used to allow a restart dump to be read regardless of the differences between the array dimensions and EDITOR definition settings in the new executable (that which is reading the restart dump) and the old executable (that which created the restart dump). Thus, it is possible, for example, to resume an MHD run without the `MHD` definition set (and thus resume the calculation hydrodynamically), or to read the inner eighth of a 64^3 data volume into any part of a new 100^3 grid, or whatever. This added flexibility of the restart dump format is new to this version of the code. Note that restart dumps generated by previous versions of the code cannot be read by `zeus32` and vice versa.

EDITOR definition:	none
logical unit:	`iodmp`
namelist:	`rescon`
filename:	`zr`*nn*`id`, where `zr` is the common prefix to all restart dumps, *nn* is a two digit integer distinguishing the multiple dumps created during a run, and `id` is a two character, user-specified problem tag.
format:	binary, one word (8 bytes Cray, 4 bytes Convex) per datum

2. 1-D NCAR PLOT FILES – These are metacode files each of which contains the 1-D plots along one of the specified 1-D slices through all of the selected variables. If, for example, m slices are specified for n variables, then each time 1-D plots are required, m metafiles will be created each containing n plots. (Note: place the command "`setenv NCARG_GKS_OUTPUT = GMETA`" in your `.login` file to ensure that the default name for metacode files is `GMETA`).

EDITOR definition: `PLT1D`

logical unit: `ioplt1`
namelist: `plt1con`
filename: `zp`*nn*`id`.*mm*, where `zp` is the common prefix to all 1-D plot files, *nn* and `id` are as defined for restart dumps, and *mm* is an extension indicating the slice number.
format: metacode – use `cgmtrans` or `plt` to read the metafiles

3. 2-D NCAR PLOT FILES – These are metacode files each of which contains the 2-D plots (contours and/or vectors) on one of the specified 2-D slices through all of the selected variables. If, for example, m slices are specified for n variables, then each time 2-D plots are required, m metafiles will be created each containing n plots.

EDITOR definition: `PLT2D`
logical unit: `ioplt2`
namelist: `plt2con`
filename: `zq`*nn*`id`.*mm*, where `zq` is the common prefix to all 2-D plot files, *nn* and `id` are as defined for restart dumps, and *mm* is an extension indicating the slice number.
format: metacode

4. 2-D PIXEL DUMPS – Each file contains a binned 2-D slice through the data volume of a single variable designed for visualization. They can be written in either raw format (one byte per datum) or HDF (four bytes per datum; see below). The raw format files can be read by NCSA IMAGETOOL and are not intended for quantitative analysis since the dynamic range (256) is too small for most purposes other than qualitative rendering. The HDF files may be read by IMAGETOOL1.1, NCSA IMAGE (open windows), or any other software package capable of reading HDF files and may be used quantitatively. In this version, polar plots are rebinned to a Cartesian plane, and dumped as Cartesian pixel plots. Thus, POLAR IMAGETOOL is no longer necessary. Because the data files are so small (especially the raw format), enough images can be written to disc during the simulation to create a smooth temporal animation of the calculation for a number of variables. Multiple slices can be specified for each variable and, in a post-processing session using NCSA DATAVU (a program which formats and annotates frames for an animation), reassembled in their proper 3-D perspective. Note that raw pixel dumps have no header. Thus, the dimensions of the dumps (needed to read the raw dumps correctly) are noted in the message log file (see below) each time a dump is created.

EDITOR definition: `PIX`
logical unit: `iopix`
namelist: `pixcon`
filename: `zi`******`nnnid`.*mm*`.h`, where `zi` is the common prefix to all 2-D pixel dumps, `**` is a two-character representation of the variable (see Table 4.1 at the end of this section), *nnn* is a 3 digit integer distinguishing the multiple dumps created during a run, `id` is the character*2 problem tag specified in `rescon`, *mm* is an extension indicating the slice number, and `h` is an extension added *only* for

HDF files.

formats: raw, one byte per datum
HDF, four bytes per datum

5. 3-D VOXEL DUMPS – Each file contains a 3-D dump of a single variable rebinned to a Cartesian grid using either raw format (one byte per datum) or HDF (four bytes per datum). These are the 3-D analogues of the 2-D pixel dumps and can be used by a variety of software packages including DATAVU and Spyglass DICER. In this release, voxel dumps may be generated in both Cartesian (`XYZ`) and cylindrical (`ZRP`) coordinates. Storing enough of these images to create a smooth 3-D animation of a run is possible, but strains the current disc space limitations. As much as 4 Megabytes per raw-format image may be required for a one million zone simulation. Note that the maximum dimensions of a voxel dump are `in-1, 2*jn-1, 2*kn-1`. Since raw voxel dumps have no header, software reading these dumps will require their dimensions as input. These are noted in the message log file as the voxel dumps are created.

EDITOR definition: `VOX`
logical unit: `iovox`
namelist: `voxcon`
filename: `zv**`*nnn*`id.h`, where `zv` is the common prefix to all 3-D voxel dumps, `**`, *nnn*, `id`, and `h` are as defined for pixel dumps.
formats: raw, one byte per datum
HDF, four bytes per datum

6. HDF FILES – These files contain 3-D data of one or more variables in the HDF format developed at NCSA, and differs from the voxel HDF dumps in that these dumps are not rebinned. The data are stored in four byte words which is more than adequate for quantitative graphical study. Most graphical software packages at NCSA use this format for data dumps. HDF files are useful because they contain header information which includes array dimensions, extrema of data, and the grid coordinates. It is highly recommended that this format be used if any post-processing of the data is to be performed using software developed outside NCSA. The size of an HDF file containing a single variable is the number of active zones times 4 bytes. For a "total" dump (all primary variables to the same HDF file), the size is the number of active zones times 32 bytes for MHD runs, or times 20 bytes for HD runs.

EDITOR definition: `HDF`
logical unit: none
namelist: `hdfcon`
filename: `zh**`*nnn*`id`, where `zh` is the common prefix to all HDF files, `**`, *nnn*, and `id` are as defined for pixel dumps.
format: HDF, four bytes per datum

7. TIME SLICE DUMPFILES – There are two types of time slice dumps, and either, both, or neither may be selected. The first is a single ascii file which contains values of various scalars at specified time intervals. The second is a metafile containing 1-D plots of

these scalars plotted as a function of time. The user selects the time interval for the ascii and metafile dumps independently. The scalars include various integral quantities such as total mass, angular momenta, magnetic monopoles, energy, *etc.*, as well as extrema of quantities such as density, pressure, divergence of magnetic field, *etc.* The user may wish to add other scalars to this format (subroutines `tslice` and `tslplot`).

EDITOR definition:	`TIMESL`
logical units:	`iotsl` and `iotslp`
namelist:	`tslcon`
filenames:	`zt`*ll*`id` (ascii file), where `zt` is the common prefix to all time slice ascii files, *ll* is incremented by one each time the job is restarted, and `id` is as defined for restart dumps. `ztp`*ll*`id` (metafile), where `ztp` is the common prefix to all time slice metafiles, *etc.*
formats:	ascii metacode

8. DISPLAY DUMP FILE – This is a single ascii file (maximum of 80 characters per line) which contains a quantitative display (matrix format) of a specified portion of various 2-D slices through any of many variables at evenly spaced time slices during a simulation. The data are scaled and converted to integers before being written to the ascii file. The dynamic range of the scaled data depends on the specified "width" of the field of view (no more than 38), and ranges from 10^2 to 10^6. For very small widths (≤ 8), the data are not scaled and written as real numbers, with three or four significant figures. This utility is much like PRTIM in AIPS, for those familiar with the Astronomical Image Processing System. Its primary use is in debugging, or when one needs to view a small portion of data quantitatively and simultaneously.

EDITOR definition:	`DISP`
logical unit:	`iodis`
namelist:	`discon`
filename:	`zd`*ll*`id`, where `zd` is the common prefix to all display files, *ll* is as defined for time slice dumps, and `id` is as defined for restart dumps.
format:	ascii

9. 2-D RADIO DUMPS – These files are similar to the 2-D pixel dumps, but contain line-of-sight integrations of various quantities rather than 2-D slices through the data volume. In this release, RADIO dumps are possible in both Cartesian (`XYZ`) and cylindrical (`ZRP`) coordinates. The integrands are all scalars (density, pressure, internal energy, magnetic pressure, velocity shear, velocity divergence, and three Stokes emissivities) and are integrated using a very fast binning algorithm, as opposed to the slower (by a factor of ~ 50) direct ray-tracing algorithm used in the original RADIO code. Files may be dumped in either raw format (one byte per datum) or HDF (four bytes per datum).

EDITOR definition:	`RADIO`
logical unit:	`iorad`
namelist:	`radcon`

filename:	`zR**`*nnn*`id.h`, where `zR` is the common prefix to all RADIO dumps, `**`, *nnn*, `id`, and `h` are as defined for pixel dumps.
formats:	raw, one byte per datum HDF, four bytes per datum

10. MESSAGE LOG FILE – This file contains all the messages that are written to the terminal by the code during execution. In addition, the grid and all the values of the namelist parameters specified in the file `inzeus` are dumped here. It serves as the log for the execution. (This file was formerly called `zeuslp`.)

EDITOR definition:	none
logical unit:	`iolog`, formerly `iolp`
namelist:	`none`
filename:	`zl`*ll*`id`, where `zl` is the common prefix to all log files, *ll* is as defined for time slice dumps, and `id` is as defined for restart dumps.
format:	ascii

11. USERDUMP – This is an EDITOR alias available for the user to include their own special type of I/O which may be desired in addition to those currently available. See Section VI for details on how to add subroutines to the code.

EDITOR definition:	none
logical unit:	`iousr`
namelist:	`usrcon`
filename:	`zu`*nn*`id`, where `zu` is the common prefix to all user dump files, *nn* and `id` are as defined for restart dumps
format:	chosen by the user

The following table lists the two-character variable representations [corresponding to the double asterisks (**) above] used for generating the filenames for pixel (P), voxel (V), HDF (H), and RADIO (R) dumps. These two-character representations are identical to those used to specify the variables to be dumped (see `pixvar` in namelist `pixcon`, `voxvar` in namelist `voxcon`, `hdfvar` in namelist `hdfcon`, and `radvar` in namelist `radcon`, Appendix 2) with the exception that variables specified by a single character (*e.g.*, `d`) appear with a trailing underscore (*e.g.*, `d_`) in the dump file name. The third column indicates the I/O types in which the variable may be dumped.

Table 4.1
Two Character Variable Representations

**	Variable	Dumps	**	Variable	Dumps
a1	1-vector potential	P	v_	velocity norm (speed)	PVH
a2	2-vector potential	P	v1	1-velocity	PVH
a3	3-vector potential	P	v2	2-velocity	PVH
b1	1-magnetic field	PVH	v3	3-velocity	PVH
b2	2-magnetic field	PVH	vv	velocity divergence	PV
b3	3-magnetic field	PVH	A_	pol'n position angle	
d_	density	PVH	AV	pola with pol'n vectors	
e_	internal energy	PVH	F_	fractional pol'n	
gp	gravitational potential	PVH	FV	fpol with pol'n vectors	R
j_	current density norm	V	I_	total intensity	R
j1	1-current density	PVH	IV	toti with pol'n vectors	
j2	2-current density	PVH	P_	polarized intensity	
j3	3-current density	PVH	PV	poli with pol'n vectors	
m_	Mach number	PVH	V_	pol'n vectors (black)	
p_	thermal pressure	PVH	VR	pol'n vectors (white)	
pb	magnetic pressure	PVH	D_	density	R
pt	thermal + magnetic pres.	PVH	E_	internal energy (pres.)	R
s1	1-momentum	PVH	PB	magnetic pressure	R
s2	2-momentum	PVH	SE	sp. int. energy (temp.)	
s3	3-momentum	PVH	SH	scalar velocity shear	
se	specific internal energy	PVH	VV	velocity divergence	
to	all field arrays	H			

V INTERACTING WITH ZEUS-3D

During interactive execution (as opposed to batch execution), the user may probe ZEUS-3D for its status, change a few input parameters, and even submit instructions to create a dump, stop, pause, resume, *etc.* This is done by typing one of many recognized three-character "interrupt messages" followed by a carriage return. Once every cycle (or time step), ZEUS-3D "glances" at the terminal buffer. If an interrupt message has been typed followed by a carriage return, ZEUS-3D will carry out the instruction to the best of its ability. If no interrupt message is found, execution will proceed without pause. Below is a list of the interrupt messages currently recognized by ZEUS-3D, along with a brief description of their function. Only the first three characters of each command (those in `typewriter` font) need be entered. Note that there are several synonyms for a number of the commands, which are separated by commas.

Controlling execution:

- `time`, `cycle`, `status`, `t`, `n`, `?`
  prints a time and cycle report, then resumes execution
- `quit`, `abort`, `crash`, `break`
  immediate emergency termination, no final dumps are made
- `stop`, `end`, `exit`, `finish`, `terminate`
  clean stop – all final dumps are made
- `halt`, `pause`, `wait`, `interrupt`
  halt execution and wait for a message from the crt or controller.
- `restart`, `go`
  restarts execution after a halt
- `tlimit`, `tfinish` (followed by a real number)
  resets the physical (problem) time
- `nlimit`, `nfinish` (followed by an integer)
  resets the cycle limit
- `ttotal`, `tcpu` (followed by an integer number of seconds)
  resets maximum cpu time to consume.
- `tsave`, `treserve` (followed by an integer number of seconds)
  resets the save time reserved for cleanup and termination

Controlling data output:

- `dump`
  creates a restart dump at current time
- `dtdmp` (followed by a real time interval)
  resets the problem time interval between restart dumps

- `pl1`
  creates a 1-D plot at current time
- `dt1` (followed by a real time interval)
  resets the problem time interval between 1-D plots
- `pl2`
  creates a 2-D plot at current time
- `dt2` (followed by a real time interval)
  resets the problem time interval between 2-D plots
- `pix`el
  creates a pixel dump at current time
- `dtpix` (followed by a real time interval)
  resets the problem time between pixel dumps
- `vox`el
  creates a voxel dump at current time
- `dtvox` (followed by a real time interval)
  resets the problem time between voxel dumps
- `usr`
  creates a user dump (calls USERDUMP) at current time
- `dtusr` (followed by a real time interval)
  resets the problem time between user dumps
- `hdf`
  creates an HDF dump at current time
- `dth` (followed by a real time interval)
  resets the problem time between HDF dumps
- `tsl`ice
  adds a time slice dump at current time to time slice file
- `dtts`lice (followed by a real time interval)
  $> 0 \Rightarrow$ resets the problem time between time slice ascii dumps
  $< 0 \Rightarrow$ resets the problem time between time slice plot dumps
- `dis`play
  adds a display dump at current time to display dump file
- `dtdis`play (followed by a real time interval)
  resets the problem time between display dumps
- `rad`io
  creates a radio dump at current time
- `dtra`dio (followed by a real time interval)
  resets the problem time between radio dumps

VI ADDING SOURCE CODE TO ZEUS-3D

6.1 ADDING AN ENTIRE SUBROUTINE

Adding source code to the ZEUS-3D package is not as difficult as one might anticipate, especially if all of the changes are restricted to the addition of entirely new subroutines. Below is a template for a subroutine called `myprob` which can be used to create a problem generator. An electronic copy of this template resides in the AFS directory:

`/afs/ncsa/projects/lca/codes/zeus3d/source`

The style is that which is used for all subroutines currently in `zeus32`.

```
*insert zeus3d.9999
*deck myprob
c=======================================================================
c
c                   B E G I N   S U B R O U T I N E
c                             M Y P R O B
c
c=======================================================================
c
       subroutine myprob
c
c    abcd:zeus3d.myprob <----------------------- initializes my problem
c                                                        september, 1990
c
c    written by:  A Busy Code Developer
c    modified 1:
c
c  PURPOSE:  Initializes all the flow variables for my problem.  More
c  description of my problem can go here.
c
c  LOCAL VARIABLES:
c
c  EXTERNALS:  BNDYFLGS, BNDYALL, BSETMAG
c
c-----------------------------------------------------------------------
c
*call rouse
       integer       i       , j       , k
       real          da      , db      , ea      , eb      , v1a
     1             , v1b     , v2a     , v2b     , v3a     , v3b
     1             , b1a     , b1b     , b2a     , b2b     , b3a
     1             , b3b
c
       real          array1d (ijkn)
       real          array2d (ijkn,ijkn)
       real          array3d (  in,  jn,  kn)
c
```

```
      equivalence   ( array1d , wa1d     )
      equivalence   ( array2d , wa2d     )
      equivalence   ( array3d , wa       )
c
      external      bndyflgs, bndyall , bsetmag
c
c-----------------------------------------------------------------------
c
c  Input parameters:
c
c    da , db    values for density
c    ea , eb    values for internal energy
c    v1a, v1b   values for 1-velocity
c    v2a, v2b   values for 2-velocity
c    v3a, v3b   values for 3-velocity
c    b1a, b1b   values for 1-magnetic field
c    b2a, b2b   values for 2-magnetic field
c    b3a, b3b   values for 3-magnetic field
c
      namelist / pgen     /
     1               da      , db      , ea      , eb      , v1a
     1             , v1b     , v2a     , v2b     , v3a     , v3b
     1             , b1a     , b1b     , b2a     , b2b     , b3a
     1             , b3b
c
c      Default values
c
      da  = 1.0
      db  = 0.1
      ea  = 0.9
      eb  = 9.0
      v1a = 0.0
      v1b = 1.0
      v2a = 0.0
      v2b = 1.0
      v3a = 0.0
      v3b = 1.0
      b1a = 0.0
      b1b = 0.0
      b2a = 0.0
      b2b = 0.0
      b3a = 0.0
      b3b = 0.0
c
      read  (ioin , pgen)
      write (iolog, pgen)
c
c      Set field arrays
c
      do 30 k=ksmn,kemx
        do 20 j=jsmn,jemx
          do 10 i=ismn,iemx
            d (i,j,k) = da
```

```
              v1(i,j,k) = v1a
              v2(i,j,k) = v2a
              v3(i,j,k) = v3a
*if -def,ISO
              e (i,j,k) = ea
*endif -ISO
*if def,MHD
              b1(i,j,k) = b1a
              b2(i,j,k) = b2a
              b3(i,j,k) = b3a
*endif MHD
10          continue
20        continue
30      continue
*if -def,ISYM
c
c       Set inflow boundary values.
c
        do 50 k=ksmn,kemx
          do 40 j=jsmn,jemx
            niib (j,k) = 3
            diib1 (j,k) = db
            v1iib1(j,k) = v1b
            v2iib1(j,k) = v2b
            v3iib1(j,k) = v3b
*if -def,ISO
            eiib1 (j,k) = eb
*endif -ISO
*if def,MHD
            b1iib1(j,k) = b1b
            b2iib1(j,k) = b2b
            b3iib1(j,k) = b3b
*endif MHD
40        continue
50      continue
*endif -ISYM
c
c        Set all boundary values
c
        call bndyflgs
        call bndyall
*if def,MHD
        call bsetmag
*endif MHD
c
        write (iotty, 2010)
        write (iolog, 2010)
2010    format('MYPROB  : Initialization complete.')
c
        return
        end
c
c=======================================================================
```

```
c
c                     E N D   S U B R O U T I N E
c                             M Y P R O B
c
c=======================================================================
c
```

There are many ingredients to this template which warrant discussion. In order of appearance, these are:

1. Ignoring for the moment the EDITOR statement `*insert zeus3d.9999`, the first line of each subroutine must be an EDITOR `*deck` (`*dk` for short) statement. Without this statement, the precompiler won't put the subroutine into a separate file, inhibiting the debugger should it be necessary. It is easiest, although not necessary, to give the deck the same name as the subroutine.

2. Note that there is no parameter list in the subroutine statement. A parameter list is unnecessary since all variables that need to be used and/or set are accessible via the common blocks. In fact, using a parameter list would inhibit the inclusion of a user-supplied subroutine using the present structure of the code.

3. All of the important variables declared in `zeus32` are in common blocks, and can be included into a subroutine simply by inserting the EDITOR statement `*call rouse` just before the local declarations are made. The EDITOR `*call` (`*ca` for short) statement is much like the VMS INCLUDE facility whereby a section of code known as a "common deck" (called `rouse` in this case) is inserted at the location of the `*call` statement. Every variable of any possible interest is declared in `rouse`, including many that the user would never need. (A description of the most widely used variables is given in Appendix 3.) At the beginning of `rouse` is an "`implicit none`" statement, which requires that the attributes of all variables used in the subroutine be declared. Note that should the user inadvertently try to use a variable name already declared in `rouse`, the compiler will flag the repetition and abort compilation. While the "`implicit none`" statement no longer requires that all externals called by the program unit be declared in an `external` statement (cft77.5.0 and later), it is still good practice to do so. In fact, if undeclared externals appear inside a nested do-loop construct, this may inhibit EDITOR's autotasking feature which microtasks `zeus32` for parallel processing on either of the Crays.

4. Should one dimensional arrays be required to store data at each grid point along one of the axes, it is best to declare the 1-D vector with dimension `(ijkn)`, as done in the template. The parameter `ijkn` is declared in `rouse` and is defined as the largest of `in`, `jn`, and `kn` (the dimensions of the 3-D arrays), also declared in `rouse`. So that no additional memory is occupied by this local array, it can be equivalenced to one of the 26 1-D scratch arrays declared in `rouse`, as done in the template. The names of the 1-D scratch arrays are `wa1d` through `wz1d`, where the "w" stands for "worker". Similarly, if a local 2-D array is required, one can avoid requiring extra memory by declaring the 2-D array with dimensions `(ijkn,ijkn)` and equivalencing it to one of eight 2-D

scratch arrays (wa2d through wh2d). Finally, a local 3-D array should be declared with dimensions (in,jn,kn) and equivalenced to one of six 3-D scratch arrays (wa through wf). Note that when one of the symmetry macros (ISYM, JSYM, KSYM) has been defined in zeus32.mac, wa2d through wf2d are equivalenced to wa through wf in rouse while wg2d and wh2d are not declared. Further, if ISYM is defined, 2-D arrays should be dimensioned (jn,kn), if JSYM is defined, 2-D arrays should be dimensioned (kn,in), and if KSYM is defined, 2-D arrays should be dimensioned (in,jn).

5. The namelist pgen is reserved for the namelist in the Problem GENerator. Of course, any name other than pgen could be used, so long as it is not already used in the input deck inzeus and the new name for the namelist is substituted for pgen in inzeus. Note that default values for the input parameters can be assigned before the namelist is read.

6. Loop 30 is a typical way the 3-D field variables (d = density, e = internal energy per unit volume, *etc.*) are assigned values. In this very simple case, the variables are assigned to the scalars read from the namelist. Note that all variables pertaining to the energy (e, eiib1, *etc.*) should be considered as energy per unit *volume* and not energy per unit *mass*. Appendix 3 has a list of all the variable names and their dimensions. The do-loop indices declared in rouse are all assigned values in the subroutine nmlsts (see Appendix 1) and so they can be used explicitly in any user-supplied subroutine called thereafter. Thus, the index for loop 30 (k) ranges from ksmn (k-start minimum) to kemx (k-end maximum). Similarly for the indices of loops 20 (j) and 10 (i). Note the use of the EDITOR *if define, *endif (*if def, *ei for short) structure which conditionally includes or excludes a segment of coding depending on whether, in this case, MHD was defined during precompilation. Similar conditionals can be based on the "truth" of any EDITOR definition, and on how aliases are set. For example, one could place an EDITOR *if alias PROBLEM.eq.myprob just after the subroutine statement, and the matching *endif just before the return statement. In this way, the subroutine would be empty (nothing between the subroutine and return statements) unless the EDITOR alias PROBLEM were set to myprob. This would prevent it from being compiled when it is not needed.

7. Loop 50 illustrates how inflow boundary values (to be applied only to those boundary zones where matter is flowing into the grid in a known fashion) can be easily set. In this case, the "inner-i-boundary" (iib) values of the flow variables are being initialized. Alternatively, one could set the inflow boundary values using the namelists iib, oib, *etc.* (see Appendix 2). Note the use of the EDITOR *if define, *endif construct to prevent this loop from being compiled in the event that ISYM is defined. If ISYM has been defined, the variables niib, *etc.* are *not* declared in rouse. Variables that are conditionally declared (depending on which EDITOR definitions are set) are noted in Appendix 3.

8. After loop 50, all the boundary values of the 3-D field arrays can be initialized by calling the subroutines bndyflgs, which sets all the secondary boundary flags according to the values set for the primary flags (niib, *etc.*), bndyall, which sets all the hydrodynamical variable boundaries, and bsetmag, which sets all the magnetic

variable boundaries. The boundaries are set depending on the values of the boundary flags (`niib`, *etc.*), which indicate the boundary type (Appendix 2). Note that the user's problem generator *must* initialize the boundary zones in addition to the active zones. If the subroutines `bndyall` and `bsetmag` are insufficient for this purpose, the boundary zones, should be set explicitly.

9. Finally, if desired, the user can write various messages to the terminal (logical unit `iotty`) or to the message log file (logical unit `iolog`). Both `iotty` and `iolog` are declared in `rouse` and set by the subroutine `mstart`.

Once the subroutine is written, it should be placed in its entirety into the change deck `chgz32`. Upon its first pass (the merge step), the preprocessor will, in this case, insert the user's subroutine into `zeus32` immediately after line 9,999 of the main program `zeus3d` (by virtue of the EDITOR statement: `*insert zeus3d.9999` appearing at the top of the subroutine template). Since `zeus3d` doesn't have 9,999 lines, EDITOR will simply stick the subroutine after the last line of the main program. It doesn't matter where in `zeus32` the subroutine gets inserted so long as it isn't in the middle of an existing deck. Immediately after the main program is as good of a place as any. Upon the second pass, the precompiler will find the user's deck and treat it as it would any other it encounters. Thus, if there are any EDITOR commands in the user's deck (such as `*call rouse`, `*if define,MHD`), they will be carried out and then expunged from the working copy of the source code. The user's subroutine will then be placed in its own file in the directory `zeus3.2`, and the name of the subroutine will be included in the makefile `makezeus`, which will then compile the subroutine and link it with the rest of the object files and libraries. Provided the EDITOR alias `PROBLEM` has been set to `myprob` in the macro file `zeus32.mac`, the user's problem generator will be called at the appropriate time during execution. Similarly, if the subroutine should be called at the location of any of the other available "plugs" in the code, set the appropriate alias (*i.e.* `SPECIAL`, `SPECIALSRC`, `SPECIALTRN`, `USERDUMP`, `PROBLEM`, `PROBLEMRESTART`, or `FINISH`; see Subsection 3.2.2 and the ZEUS-3D skeleton in Appendix 1) in `zeus32.mac` to the subroutine name.

6.2 MICROSURGERY USING EDITOR

It is possible for the user to alter individual lines of code in `zeus32` without actually changing the original source code. In this way, the changes made can be kept separate from the code, and thus not lost in the abyss of `zeus32`. In addition, the user's changes could, in principle, be incorporated into the master code at a later date and become part of the next release. To do this, there are two things required: an EDITOR listing of the code and a short tutorial on how to use EDITOR. For those who have worked with HISTORIAN, all this should seem very familiar. For those who haven't, take heart – the structure is very intuitive. However, one must ensure that the changes made don't break something else in the code.

To get an EDITOR listing of the code, run the following script file (call it `number.s`) by typing `csh -v number.s`

```
#============ SCRIPT FILE TO CREATE A NUMBERED LISTING ==============#
#
#
#======================> Create the input deck for EDITOR, and execute.
\rm -f inedit
cat << EOF > inedit
 \$editpar   inname='zeus32'
          , ibanner=1, job=1, inumber=3, itable=1, ixclude=1          \$
EOF
xedit21
```

An electronic copy of this and other useful scripts reside on landrew in:

`/afs/ncsa/projects/codes/zeus3d/editor/source`

This script file will run EDITOR in its numbering mode, and produce a listing with a table of contents, and various labels on each line. The numbered file will be called `zeus32.n`, and can be printed with a printer capable of 132 column output. Beware that at 60 lines per page, there will be some 750 pages of output! The third column to the right of the source listing is the number of lines since the most recent EDITOR `*deck` or `*cdeck` statement. This is the column needed to perform microsurgery on the master file.

During preprocessing, EDITOR makes two major passes over the code. The first pass does the merging of the change deck `chgzeus` (which contains `zeus32.mac` and `chgz32`) into the main code. EDITOR commands performed during this pass include:

1. `*insert` *deckname.n* – inserts text immediately following the `*insert` command into the source code directly after line n in deck (or cdeck – common deck) *deckname*. The value of n is determined from the third column to the right of the source code in the numbered listing, `zeus32.n`.
2. `*delete` *deckname.n,m* – deletes lines n through m in deck (or cdeck) *deckname*, and replaces it with the text immediately following the `*delete` command, if any. Note that m must be greater than n. If m is missing altogether, then $m = n$ will be assumed.

That's it. An example:

```
*delete zeus3d.10,20
       a = b
       b = c
*insert mstart.100
       d(i,j,k) = 1.0
*i zeus3d.100
       c = d
*d zeus3d.120
```

Note that `*d` and `*i` are short forms for `*delete` and `*insert` respectively. In addition,

`*replace` (`*rp` for short) is a synonym for `*delete`. In the example, lines 10 through 20 in the main program `zeus3d` are replaced with the two lines which set `a` and `b`, a single line setting `d(i,j,k)` is inserted after line 100 in subroutine `mstart`, a single line setting `c` is inserted after line 100 in `zeus3d`, and line 120 in `zeus3d` is simply deleted. These statements should be placed in the file `chgz32`, and would be incorporated into the master code during the first pass of the preprocessing.

To aid the user in deciding what changes to make and where to make them, a flow chart showing the sequence of subroutine calls in ZEUS-3D is given in Appendix 1. This will be particularly useful once faced with the task of comprehending the source code listing, `zeus32.n`.

If EDITOR detects any merge syntax errors or conflicts during the merge, it will write the merged file (as best as could be done given the error(s) detected) into a file named `zeus32.m` and insert an error message immediately after each offending line. A merge error will prevent the second pass of preprocessing (*i.e.*, precompilation) from being executed and the user will be told what character pattern to search for in the file `zeus32.m` in order to find the generated error messages.

Should the merge step be successful, EDITOR goes through a second pass and performs all the precompilation commands. These include:

1. `*if define,`*macro* – the following source code is kept provided the macro is defined by a `*define` statement somewhere in the file.
2. `*if -define,`*macro* – the following source code is kept provided the macro is *not* defined by a `*define` statement somewhere in the file.
3. `*if def,.not.`*macro* – same as 2. Note that `def` is an acceptable short form for `define`.
4. `*if def,`*macro1*`.and.`*macro2* – the following source code is kept provided both macros are defined by a `*def` statement somewhere in the file.
5. `*if def,`*macro1*`.or.`*macro2* – the following source code is kept provided either macro is defined by a `*def` statement somewhere in the file.
6. `*if alias` *macro*`.eq.`*phrase* – the following source code is kept provided the alias *macro* has been set to the character string *phrase* by an `*alias` statement somewhere in the file.
7. `*if alias` *macro*`.ne.`*phrase* – the following source code is kept provided the alias *macro* has *not* been set to the character string *phrase* by an `*alias` statement somewhere in the file.
8. `*else` – the following source code is kept if the source code following the previous `*if` (and all the way to this `*else` statement) was not kept, *i.e.* if the truth value of the previous `*if` is false.
9. `*endif` – closes the previous `*if`, `*else` structure. All source code following the `*endif` statement is not affected by the previous `*if` or `*else` statements. For every `*if` statement, there must be an `*endif` statement which follows.

10. `*call` *deckname* – includes the contents of the common deck *deckname* at the location of the `*call` statement.

These precompiler commands can be used to construct the changes to be inserted into `zeus32` using the EDITOR `*delete` and `*insert` commands. All changes should be placed in the file `chgz32`. Note that during both passes, the `*deck` and `*cdeck` statements are used as reference points, and are then expunged from the source code during the second pass. If any precompilation syntax errors are detected, EDITOR will write the precompiled file (as best as could be done given the error(s) detected) into a file named `zeus32.f` and insert an error message immediately after each offending line. EDITOR will abort further processing (namely splitting up the source code into separate files for each subroutine, substituting `namelist` statements with subroutine calls, autotasking) and the user will be told what character pattern to search for in the file `zeus32.f` in order to find the generated error messages. On the other hand, if the precompilation is successful, EDITOR will update the files in the directory `zeus3.2`. The makefile `makezeus` will then compile only those subroutines affected by the changes made, and the executable will be created.

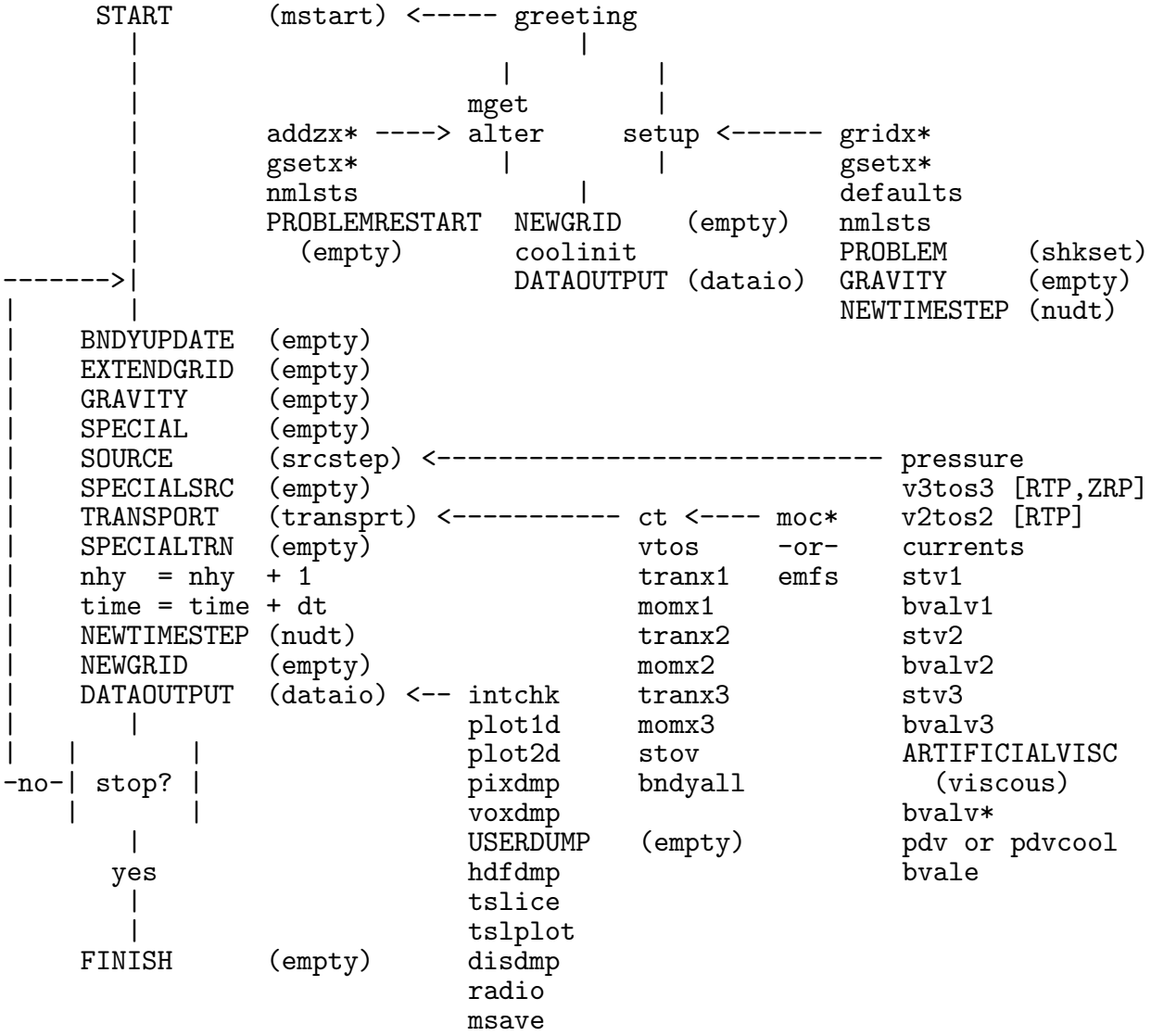
VII QUICK SUMMARY

This final section is intended to serve as a quick reference sheet for those who are already familiar with running ZEUS-3D.

1. Set the macros in `zeus32.mac` (Subsection 3.2, Appendix 1).
2. Make the necessary changes to the parameters in the change deck `chgzeus` (Segment 3.3.3) and the input parameters in the input deck `inzeus` (Segment 3.3.5 and Appendix 2).
3. Put the desired source code changes, if any, into the file `chgz32` (Section VI, Appendices 2 and 3).
4. Run the script file to create the ZEUS-3D executable by typing `zeus32.s` (assuming you already have execute permission)
5. Run the executable by either typing `xzeus32`, or by submitting the job to the appropriate batch queue (Subsection 3.4)

APPENDIX 1: THE ZEUS-3D SKELETON

Modules in upper case are EDITOR aliases, set in `zeus32.mac`. Modules in lower case are actual subroutine names in the source code. As asterisk (*) in a subroutine name is a "wildcard" for `1`, `2`, and `3`. Exemplary choices for the EDITOR aliases (enclosed parenthetically) have been made. These choices are appropriate for the Sod shock tube problem. All the existing choices for the EDITOR module name aliases follow on the next page.



START	mstart	
EXTENDGRID	empty	
	extend	
BNDYUPDATE	empty	
	breset	to reset flow-in boundary values, used in test problems
	wiggle	to wiggle jet inlet
	bgen	to generate magnetic field at jet inlet
	jetbndy	calls both subroutines `wiggle` and `bgen`
GRAVITY	empty	no self-gravity
	gravity	DADI algorithm to update self-gravitational potential
SPECIAL	empty	
		user-defined module for additional physics
SOURCE	empty	for advection tests
	srcstep	
SPECIALSRC	empty	
		user-defined module for additional source terms
TRANSPORT	empty	
	transprt	
SPECIALTRN	empty	
	resetv	for advection tests
		user-defined module for additional transport terms
NEWGRID	empty	no grid velocity
	newgrid	
NEWTIMESTEP	nudt	full dynamics
	advectdt	for advection tests
DATAOUTPUT	empty	
	dataio	
FINISH	empty	
		user-defined module called once at the end of execution
USERDUMP	empty	
		user-defined I/O module
ARTIFICIALVISC	empty	
	viscous	von Neumann-Richtmyer artificial viscosity
PROBLEM	advect	for advection tests
	blast	for explosions
	homol	
	shkset	for shock tube tests
	zpinch	Bennett pinch problem
	mhdrot	aligned magnetic rotator test
	jetinit	initializes jet simulations
		user-defined module to initialize flow variables
PROBLEMRESTART	empty	
	resetb	sets all magnetic field variables to zero
		user-defined module to alter variables for restarted job

APPENDIX 2: THE NAMELISTS

There are nearly 500 namelist parameters, and one might be overwhelmed at the prospect of setting values for every one. Take heart – many of the defaults can be used for most applications. As a good start, use the input deck given in the `zeus32.s` template (Subsection 3.3), and then alter or add the appropriate parameters.

On the next page begins a complete catalogue of all the input parameters in `zeus32`. The parameters are grouped together in "namelists" and discussion for each namelist is contained within a segment headed by the name of the namelist and the subroutine in which the namelist is called. For example, the first namelist is `iocon` (input/output control) and is called by the subroutine `mstart`. After each heading is a discussion of what the namelist controls, a list of all the parameters which are elements of the namelist, and finally the syntax used in `zeus32` to declare the namelist.

For the uninitiated, a "namelist" is a standard feature of FORTRAN9X and a non-standard feature of most FORTRAN77 compilers which provides a very convenient way to specify input data. Historically, however, the UNIX/UNICOS namelists were fraught with inconveniences, and so the preprocessor EDITOR was built with a namelist emulator which resembles the namelist under the now all-but-extinct CTSS operating system. The following discussion, therefore, reflects some of the syntactic rules appropriate to the EDITOR namelist, and not necessarily those of the UNIX/UNICOS namelist.

In order to specify an input parameter, one merely needs to set it to the desired value as done in the input deck `inzeus` found in the sample script file `zeus32.s` (Subsection 3.3). The order in which the variables are declared in the namelist declaration need not be adhered to in the input deck nor must all of the variables be set. So long as the variable specified in the input deck is an element of the namelist, then the namelist facility will set the variable to the specified quantity.

There are a few rules to bear in mind. The namelists in the input deck must be in the same order as they are encountered during execution. If no parameters are to be set, an empty namelist (one with the namelist name between two $ sentinels) must be put in the correct place. There is no problem with namelists appearing that are never read, but a read to a non-existent namelist will generate a namelist error message and abort execution. In this catalogue, the order of the namelists is the same as the order in which they appear in `inzeus` and the order in which they are encountered in `zeus32`. The syntactic rules of setting the variables can be gleaned from the input deck `inzeus` (Subsection 3.3). Column 1 is reserved for a '`c`' to "comment out" a namelist line. Namelist lines commented out will be echoed on the CRT when encountered in the input deck. Column 2 is reserved for the leading $ sentinel. The specification of the namelist may start in column 3 and must terminate with a second $ sentinel. Until the second $ sentinel is found, all lines will be interpreted as part of the same namelist. All characters appearing after the 72nd column will be ignored, including the closing $ sentinel, should it inadvertently be placed there.

1. IOCON – I/O CONtrol, read from subroutine MSTART

This namelist sets the logical units to be used during execution. Typically, these parameters will not need to be set to anything other than their default values. These parameters are *not* written to the restart dump. Therefore, all non-default values for any of the parameters in this namelist must be set each time the job is resumed.

parameter	description	default
iotty	logical unit for terminal (standard output)	6
ioplt1	logical unit for 1-D plots using NCAR graphics	99
ioplt2	logical unit for 2-D plots using NCAR graphics	99
iolog	logical unit for message log dump	30
iodmp	logical unit for restart dumps	31
iopix	logical unit for pixel dumps	32
iousr	logical unit for user dumps	33
iotsl	logical unit for time slice (history) ascii dumps	34
iotslp	logical unit for time slice (history) plot dumps	99
iovox	logical unit for voxel dumps	35
iodis	logical unit for display dumps	36
iorad	logical unit for RADIO dumps	37

```
WARNING: AVOID LOGICAL UNIT 3.  APPARENT CONFLICT WITH NCAR.
NOTE   : IOTTY MAY BE SET TO 6 (TO GET CRT OUTPUT) OR 0 (NO OUTPUT).

      namelist / iocon    /
     1               iotty   , ioplt1  , ioplt2  , iolog   , iodmp
     2             , iopix   , iousr   , iotsl   , iotslp  , iovox
     3             , iodis   , iorad
```

2. RESCON – REStart dump CONtrol, read from subroutine MSTART

This namelist determines if the job is to be started from initial conditions, or if it is to be restarted from a previous run. These parameters are *not* written to the restart dump. Therefore, all non-default values for any of the parameters in this namelist must be set each time the job is resumed.

The default values are set for starting from initial conditions, which occurs when the third and fourth characters in `resfile` are `00`. To restart a job, one can usually use the same input deck as was used for the original run with `resfile` set to the desired restart dump name. In addition, parameters in the namelist `pcon` may have to be changed.

The parameters `*getm?`; `*=i,j,k`, `?=n,x` are designed so that only a portion of the restart dump may be read, and/or so that the data may be read into a larger grid. That is, it is no longer necessary for the field arrays in a restarted job to be dimensioned the same as those in the run which generated the restart dump.

Example 1: For a straight restart without altering the grid or the EDITOR macros, leave the values of `igetmn`, *etc.* to their defaults and make sure that the parameters `in`, *etc.* are set to the same values as in the run which generated the restart dump.

Example 2: If the first run was on a 64^3 grid, and the user wishes to read only the inner eighth of the data and position the data at the center of a larger 100^3 grid, and if the new portion of the grid is to be determined from the existing grid, then the following settings are necessary:

```
igetmn = jgetmn = kgetmn = 17, iaddz = 1
igetmx = jgetmx = kgetmx = 48, jaddz = 1
iputmn = jputmn = kputmn = 35, kaddz = 1
```

The desired portion of the restart dump will be read and loaded into the 100^3 grid between `i=35,66`, `j=35,66`, `k=35,66`. In addition, `x1a(35:66)` (see Appendix 3 for a discussion of the naming convention for the grid variables) will be filled by the values of `x1a(17:48)` in the restart dump. The code will detect that the grids `x1a`, `x2a`, `x3a` are now incomplete, and will call the appropriate modules to add zones to the `x1-`, `x2-`, and `x3`-grids. If the user wishes, (`*addz=1`, `*=i,j,k`), the new portion of the grid may be determined automatically from the existing grid. In this example, `x1a(1:34)` would be determined (*i.e.*, `dx1min`, `x1rat`, *etc.*, see namelist `ggen1`) from `x1a(35:37)`. Similarly, `x1a(67:100)` would be determined from `x1a(64:66)`. Alternatively, the user may opt to set the new portion of the grid manually. In this case, one should set `*addz=0` and proceed with setting the namelists `ggen1`, `ggen2`, `ggen3`. (See discussion in `ggen1`.) Note that if the user selects the manual option `*addz=0`, it is imperative that the portion of the new grid that overlaps the old grid be, in fact, identical to the old grid. Next, all arrays will be padded with values at the edges of the portion read. Thus `d(1:34,j,k)=d(35,j,k)`, `d(67:100,j,k)=d(66,j,k)` (where `d` is the density array – see Appendix 3), *etc.* Of course, the user is free to overwrite the padded portion of the arrays with whatever is necessary by linking the appropriate user-supplied subroutine to the EDITOR macro `PROBLEMRESTART` (subsection 3.2.2).

Finally, a job may be resumed from a restart dump with different EDITOR macros defined or not. Thus, if a job that began with magnetic fields is to be resumed without them, the user may recompile `zeus32` *without* magnetic fields (`MHD` not defined) and then blindly read the restart dump which contains magnetic field arrays. There is enough information in the restart dump that the code can selectively read the non-magnetic part of the dump and resume the calculation as though there were never any magnetic fields. Of course, what happens dynamically once the flow no longer "feels" the magnetic fields may be another matter.

parameter	description	default
dtdmp	problem time interval between restart dumps = 0 => no restart dumps (probably a bad idea) > 0 => write each dump to a new file < 0 => overwrite old even (odd) numbered dump with new even (odd) numbered dump	0.0
id	character*2 problem tag appended to filenames	'aa'
resfile	restart dump filename	'zr00aa'
igetmn	minimum x1-index (i) to be read from restart dump	1
igetmx	maximum x1-index (i) to be read from restart dump	in
iputmn	i-index at which x1a(igetmn) is stored	1

```
iaddz      < 0 => no new zones are generated                          -1
           = 0 => call GRIDX1 to redo entire grid
           > 0 => new zone spacing determined from existing grid
```

The variables (jgetmn, jgetmx, jputmn, jaddz) and (kgetmn, kgetmx, kputmn, kaddz) are analogous to (igetmn, igetmx, iputmn, iaddz) for the 2- and 3-directions respectively.

```
      namelist / rescon   /
     1               dtdmp   , id      , resfile , igetmn  , igetmx
     2             , jgetmn  , jgetmx  , kgetmn  , kgetmx  , iputmn
     3             , jputmn  , kputmn  , iaddz   , jaddz   , kaddz
```

3. GGEN1 – Grid GENerator (for x1), read from subroutine GRIDX1

This namelist controls how the grid is determined in the 1-direction. All the parameters in this namelist, as well as those in namelists ggen2, ggen3, and those read by subroutine nmlsts are written to the restart dump. These values, therefore, will be the "default" values of the parameters for any run resumed from the restart dump.

The grid can be created all at once or in several blocks. Each block requires a separate read of this namelist specifying how that portion of the grid is to be computed. The parameter lgrid should be set to .true. only for the last block.

There are two types of gridding. The first is "ratioed gridding" where the distance across a zone is a fixed multiple of the distance across the previous zone. If this multiple is 1, then the zones are uniform. If the multiple is 1.1, then each zone is 10% larger than the previous one. If the multiple is 0.9, then each zone is 10% smaller than the previous one. To determine a block of ratioed zones uniquely, one must specify the number of zones in the block (nbl), the minimum and maximum extent of the block in coordinate units (x1min, x1max), and EITHER the smallest zone size in the block (dx1min) OR the ratio to use between zones (x1rat). Specifying either dx1min or x1rat will allow the other to be computed.

The second type of gridding is "scaled gridding" where the coordinate value is some fixed multiple of the previous coordinate value. For ratioed grids, dx(n)=mult*dx(n-1). For scaled grids, x(n)=mult*x(n-1). For example, scaled gridding would be appropriate for the r-direction in spherical polar coordinates if the zones were all to have the same *shape*. To determine a block of scaled zones uniquely, one must specify the number of zones in the block (nbl) and the minimum and maximum extent of the block in coordinate units (x1min, x1max). Neither dx1min nor x1rat are needed.

The grid can be scaled to physical units most conveniently by setting the multiplicative factor x1scale to the desired scaling value.

For restarted jobs, there is a third gridding option. Setting igrid to zero will cause the grid generator to skip over the nbl zones specified for this block. Thus, in the second example in the discussion for namelist rescon, one could set the new zones for the x1-direction manually with three ggen1 namelist "cards". The first card would set zones (1:34) in whatever manner desired with the condition that the last zone of the new grid ends where the first zone of the old grid begins. The second card would set igrid=0 and nbl=32. This would leave zones (35:66) alone since they were set when the restart dump

was read. Finally, the third card would set zones (67:100) in whatever manner desired with the condition that the first zone of the new grid begins where the last zone of the old grid ends.

Other than remaining within the memory limits of the machine, there are two practical considerations when choosing the number of zones for each of the three dimensions. First, if at all possible, the greatest number of zones should be along the 1-direction so that the vector length of the vectorized loop is as long as possible. Second, if the code is to be multitasked, specify $nN - 1$ active zones in the 3-direction, where n is a positive integer and N is the number of parallel processors available on the machine. This will yield the best overall degree of parallelism.

```
parameter                       description                              default

nbl         number of active zones in block being generated                1
x1min       x1a(imin); bottom position of block                            0.0
x1max       x1a(imax); top    position of block                            0.0
x1scale     arbitrary scaling factor for "x1min" and "x1max"                1.0
igrid       method of computing zones.                                     1
            = 0  => block has already been set (restarted runs only)
            =+1  => (ratioed) use input "x1rat" to compute "dx1min".
                    "dx1min" = size of first zone in block
            =-1  => (ratioed) use input "x1rat" to compute "dx1min".
                    "dx1min" = size of last  zone in block
            =+2  => (ratioed) use input "dx1min" to compute "x1rat".
                    "dx1min" = size of first zone in block
            =-2  => (ratioed) use input "dx1min" to compute "x1rat".
                    "dx1min" = size of last  zone in block
            = 3  => (scaled) compute "x1rat" and "dx1min" from "nbl".
x1rat       desired ratio dx1a(i+1) / dx1a(i)                              1.0
dx1min      desired difference x1a(imin+1) - x1a(imin)                     0.0
lgrid       =.false. => read another block (namelist card).              .false.
            =.true.  => all blocks are read in.  Do not look for
                        another "ggen1" namelist card.

       namelist / ggen1    /
     1               nbl     , x1min   , x1max   , x1scale , igrid
     2             , x1rat   , dx1min  , lgrid
```

4. GGEN2 – Grid GENerator (for x2), read from subroutine GRIDX2

See comments for GGEN1.

```
parameter                       description                              default

nbl         number of active zones in block being generated                1
x2min       x2a(jmin); bottom position of block                            0.0
x2max       x2a(jmax); top    position of block                            0.0
x2scale     arbitrary scaling factor for "x2min" and "x2max"                1.0
igrid       method of computing zones.                                     1
            = 0  => block has already been set (restarted runs only)
```

```
          =+1  => (ratioed) use input "x2rat" to compute "dx2min",
                  "dx2min" = size of first zone in block
          =-1  => (ratioed) use input "x2rat" to compute "dx2min",
                  "dx2min" = size of last  zone in block
          =+2  => (ratioed) use input "dx2min" to compute "x2rat",
                  "dx2min" = size of first zone in block
          =-2  => (ratioed) use input "dx2min" to compute "x2rat",
                  "dx2min" = size of last  zone in block
          = 3  => (scaled) compute "x2rat" and "dx2min" from "nbl".
x2rat     desired ratio dx2a(j+1) / dx2a(j)                          1.0
dx2min    desired difference x2a(jmin+1) - x2a(jmin)                 0.0
units     sets the angular units (character*2, RTP only)            'rd'
          'rd' => radians, 'pi' => pi radians, 'dg' => degrees
lgrid     =.false. => read another block (namelist card).        .false.
          =.true.  => all blocks are read in.  Do not look for
                      another "ggen2" namelist card.

      namelist / ggen2    /
    1              nbl     , x2min   , x2max   , x2scale , igrid
    2            , x2rat   , dx2min  , units   , lgrid
```

5. GGEN3 – Grid GENerator (for x3), read from subroutine GRIDX3

See comments for GGEN1.

```
parameter                       description                          default

nbl       number of active zones in block being generated             1
x3min     x3a(kmin); bottom position of block                         0.0
x3max     x3a(kmax); top    position of block                         0.0
x3scale   arbitrary scaling factor for "x3min" and "x3max"            1.0
igrid     method of computing zones.                                  1
          = 0  => block has already been set (restarted runs only)
          =+1  => (ratioed) use input "x3rat" to compute "dx3min",
                  "dx3min" = size of first zone in block
          =-1  => (ratioed) use input "x3rat" to compute "dx3min",
                  "dx3min" = size of last  zone in block
          =+2  => (ratioed) use input "dx3min" to compute "x3rat",
                  "dx3min" = size of first zone in block
          =-2  => (ratioed) use input "dx3min" to compute "x3rat",
                  "dx3min" = size of last  zone in block
          = 3  => (scaled) compute "x3rat" and "dx3min" from "nbl".
x3rat     desired ratio dx3a(k+1) / dx3a(k)                           1.0
dx3min    desired difference x3a(kmin+1) - x3a(kmin)                  0.0
units     sets the angular units (character*2, ZRP and RTP only)     'rd'
          'rd' => radians, 'pi' => pi radians, 'dg' => degrees
lgrid     =.false. => read another block (namelist card).        .false.
          =.true.  => all blocks are read in.  Do not look for
                      another "ggen3" namelist card.

      namelist / ggen3    /
    1              nbl     , x3min   , x3max   , x3scale , igrid
```

```
     2             , x3rat    , dx3min   , units    , lgrid
```

6. PCON – Problem CONtrol, read from subroutine NMLSTS

Determines the criteria for terminating the job.

parameter	description	default
nlim	cycles to run	0
tlim	physical (problem) time to stop calculation	0.0
ttotal	total seconds of execution time permitted for job	0.0
tsave	seconds of execution time reserved for cleanup	0.0

```
       namelist / pcon     /
     1               nlim    , tlim    , ttotal   , tsave
```

7. HYCON – HYdro CONtrol, read from subroutine NMLSTS

Set parameters which control the hydrodynamics. Default values are advisable. If `itote=0`, all energy variables should be interpreted as *internal energy density per unit volume*. Evolving the internal energy density will ensure positive definite pressures, but will introduce numerical deviations from total energy conservation which may become severe at steep gradients such as strong shocks. If `itote=1`, all energy variables should be interpreted as *total energy per unit volume*, and thus is the sum of internal, kinetic, and gravitational energy densities. Evolving the total energy density will ensure energy conservation to within machine round-off, but may yield negative pressures in extreme cases where the internal energy is a small fraction of the total energy. Using 64-bit words should prevent this from ever being a problem. To date, little has been done with the total energy option, and so it may be advisable to set `itote=0` unless one expects particularly strong shocks (Mach numbers > 100, say). Also, if either `MHD` or `ISO` is defined, `itote` will be reset to 0.

parameter	description	default
qcon	quadratic artificial viscosity (q) constant	2.0
qlin	linear artificial viscosity (q) constant	0.0
courno	Courant number	0.5
dtrat	ratio of "dtmin" to initial value of "dt"	0.001
iord	order of advection scheme used by all variables. Legal values are 1 (donor cell), 2 (van Leer), or 3 (ppa)	2
iord**	order of advection scheme to be used for variable ** 0 => "iord**" is set to "iord".	0
istp	steepener switch (third order only) for all variables. 0 => always off, 1 => always on, 2 => on only at contact discontinuity and only for density	0
istp**	steepener switch (third order only) for variable ** < 0 => "istp**" is set to "istp".	-1

```
**floor    smallest value desired for variable **              d,e   tiny
                                                               rest  0.0
icool      0 => use PDV in SRCSTEP                                    0
           1 => use PDVCOOL in SRCSTEP for pdv work with arbitrary
                cooling function
itote      0 => solve the internal energy equation                    0
           1 => solve the total energy equation

       namelist / hycon    /
     1               qcon    , qlin    , courno  , dtrat   , iord
     2             , iordd   , iords1  , iords2  , iords3  , iorde
     3             , iordb1  , iordb2  , iordb3  , istp    , istpd
     4             , istps1  , istps2  , istps3  , istpe   , istpb1
     5             , istpb2  , istpb3  , dfloor  , efloor  , v1floor
     6             , v2floor , v3floor , b1floor , b2floor , b3floor
     7             , icool   , itote
```

8. IIB – Inner I Boundary control, read from subroutine NMLSTS

This namelist specifies both the boundary type and the inflow values of all the flow variables for the inner i boundary. These variables are *not* declared if the EDITOR macro `ISYM` is set. Any one of 6 MHD boundary conditions (`nflo`) may be specified independently at every boundary zone. These boundary conditions are:

```
nflo = 1  =>  reflecting; v(normal) = b(normal) = 0
     =-1  =>  reflecting, with inversion of 3-components
     = 2  =>  flow out
     = 3  =>  flow in
     = 4  =>  periodic
     = 5  =>  reflecting; v(normal) = b(tangential) = 0
```

The boundary values for the variables are used only in the event that a zone along the boundary is inflow (`nflo=3`). Otherwise, the boundary value is determined from the flow variables on the active portion of the computational grid. The flow variables are `d` (density), `e` (internal energy per volume), `v1` (1-velocity), `v2` (2-velocity), `v3` (3-velocity), `b1` (1-magnetic field), `b2` (2-magnetic field), `b3` (3-magnetic field), `emf1` (1-emf), `emf2` (2-emf), and `emf3` (3-emf). Note that the `emf`'s are used to compute the magnetic fields, and their boundary values may be specified by the user. Instead, however, it is often easier just to specify the magnetic field and velocity components along the boundary, and then let the code determine the `emf`'s. Note that there are two sets of boundary values for most variables, and three for the tangential `emf`'s. Set all boundaries for every variable whose boundaries are to be set.

Finally, while the boundary values for the gravitational potential (`gpiib`, *etc.*) may be set here, the boundary type (`igpiib`, *etc.*) is actually set in the namelist `grvcon`. Normally, the code will determine the values for (`gpiib`, *etc.*) from the boundary type specified, and so it is not usually necessary, or even desirable, to set (`gpiib`, *etc.*) manually.

```
parameter                    description                          default

niib  (j,k)      "nflo" of inner i boundary on sweep j,k             2
**iib1(j,k)      first  inner i boundary value of variable **      **floor
                 for sweep j,k (flow in only)
**iib2(j,k)      second inner i boundary value of variable **      **floor
                 for sweep j,k (flow in only)
**iib3(j,k)      third  inner i boundary value for emf's             0.0
                 for sweep j,k (flow in only)

       namelist / iib      /
     1              niib    , diib1   , diib2   , v1iib1  , v1iib2
     2            , v2iib1  , v2iib2  , v3iib1  , v3iib2
*if -def,ISO
     3            , eiib1   , eiib2
*endif -ISO
*if def,GRAV
     4            , gpiib
*endif GRAV
*if def,MHD
     5            , b1iib1  , b1iib2  , b2iib1  , b2iib2  , b3iib1
     6            , b3iib2  , emf1iib1, emf1iib2, emf1iib3, emf2iib1
     7            , emf2iib2, emf2iib3, emf3iib1, emf3iib2, emf3iib3
*endif MHD
```

9. OIB – Outer I Boundary control, read from subroutine NMLSTS

This namelist specifies both the boundary type and the in-flow values of all the flow variables for the outer i boundary. These variables are *not* declared if the EDITOR macro ISYM is set. See comments for IIB.

```
parameter                    description                          default

noib  (j,k)      "nflo" of outer i boundary on sweep j,k             2
**oib1(j,k)      first  outer i boundary value of variable **      **floor
                 for sweep j,k (flow in only)
**oib2(j,k)      second outer i boundary value of variable **      **floor
                 for sweep j,k (flow in only)
**oib3(j,k)      third  outer i boundary value for emf's             0.0
                 for sweep j,k (flow in only)

       namelist / oib      /
     1              noib    , doib1   , doib2   , v1oib1  , v1oib2
     2            , v2oib1  , v2oib2  , v3oib1  , v3oib2
*if -def,ISO
     3            , eoib1   , eoib2
*endif -ISO
*if def,GRAV
     4            , gpoib
*endif GRAV
*if def,MHD
     5            , b1oib1  , b1oib2  , b2oib1  , b2oib2  , b3oib1
```

```
     6             , b3oib2  , emf1oib1, emf1oib2, emf1oib3, emf2oib1
     7             , emf2oib2, emf2oib3, emf3oib1, emf3oib2, emf3oib3
*endif MHD
```

10. IJB – Inner J Boundary control, read from subroutine NMLSTS

This namelist specifies both the boundary type and the inflow values of all the flow variables for the inner j boundary. These variables are *not* declared if the EDITOR macro JSYM is set. See comments for IIB.

```
parameter                    description                          default

nijb  (k,i)     "nflo" of inner j boundary on sweep k,i             2
**ijb1(k,i)     first  inner j boundary value of variable **      **floor
                for sweep k,i (flow in only)
**ijb2(k,i)     second inner j boundary value of variable **      **floor
                for sweep k,i (flow in only)
**ijb3(k,i)     third  inner j boundary value for emf's              0.0
                for sweep k,i (flow in only)

       namelist / ijb      /
     1              nijb    , dijb1   , dijb2   , v1ijb1  , v1ijb2
     2            , v2ijb1  , v2ijb2  , v3ijb1  , v3ijb2
*if -def,ISO
     3            , eijb1   , eijb2
*endif -ISO
*if def,GRAV
     4            , gpijb
*endif GRAV
*if def,MHD
     5            , b1ijb1  , b1ijb2  , b2ijb1  , b2ijb2  , b3ijb1
     6            , b3ijb2  , emf1ijb1, emf1ijb2, emf1ijb3, emf2ijb1
     7            , emf2ijb2, emf2ijb3, emf3ijb1, emf3ijb2, emf3ijb3
*endif MHD
```

11. OJB – Outer J Boundary control, read from subroutine NMLSTS

This namelist specifies both the boundary type and the inflow values of all the flow variables for the outer j boundary. These variables are *not* declared if the EDITOR macro JSYM is set. See comments for IIB.

```
parameter                    description                          default

nojb  (k,i)     "nflo" of outer j boundary on sweep k,i             2
**ojb1(k,i)     first  outer j boundary value of variable **      **floor
                for sweep k,i (flow in only)
**ojb2(k,i)     second outer j boundary value of variable **      **floor
                for sweep k,i (flow in only)
**ojb3(k,i)     third  outer j boundary value for emf's              0.0
                for sweep k,i (flow in only)
```

```
       namelist / ojb      /
     1               nojb    , dojb1   , dojb2   , v1ojb1  , v1ojb2
     2             , v2ojb1  , v2ojb2  , v3ojb1  , v3ojb2
*if -def,ISO
     3             , eojb1   , eojb2
*endif -ISO
*if def,GRAV
     4             , gpojb
*endif GRAV
*if def,MHD
     5             , b1ojb1  , b1ojb2  , b2ojb1  , b2ojb2  , b3ojb1
     6             , b3ojb2  , emf1ojb1, emf1ojb2, emf1ojb3, emf2ojb1
     7             , emf2ojb2, emf2ojb3, emf3ojb1, emf3ojb2, emf3ojb3
*endif MHD
```

12. IKB – Inner K Boundary control, read from subroutine NMLSTS

This namelist specifies both the boundary type and the inflow values of all the flow variables for the inner k boundary. These variables are *not* declared if the EDITOR macro KSYM is set. See comments for IIB.

parameter	description	default
nikb (i,j)	"nflo" of inner k boundary on sweep i,j	2
**ikb1(i,j)	first inner k boundary value of variable ** for sweep i,j (flow in only)	**floor
**ikb2(i,j)	second inner k boundary value of variable ** for sweep i,j (flow in only)	**floor
**ikb3(i,j)	third inner k boundary value for emf's for sweep i,j (flow in only)	0.0

```
       namelist / ikb      /
     1               nikb    , dikb1   , dikb2   , v1ikb1  , v1ikb2
     2             , v2ikb1  , v2ikb2  , v3ikb1  , v3ikb2
*if -def,ISO
     3             , eikb1   , eikb2
*endif -ISO
*if def,GRAV
     4             , gpikb
*endif GRAV
*if def,MHD
     5             , b1ikb1  , b1ikb2  , b2ikb1  , b2ikb2  , b3ikb1
     6             , b3ikb2  , emf1ikb1, emf1ikb2, emf1ikb3, emf2ikb1
     7             , emf2ikb2, emf2ikb3, emf3ikb1, emf3ikb2, emf3ikb3
*endif MHD
```

13. OKB – Outer K Boundary control, read from subroutine NMLSTS

This namelist specifies both the boundary type and the inflow values of all the flow variables for the outer k boundary. These variables are *not* declared if the EDITOR

macro KSYM is set. See comments for IIB.

```
parameter                    description                              default

nokb  (i,j)     "nflo" of outer k boundary on sweep i,j                  2
**okb1(i,j)     first  outer k boundary value of variable **          **floor
                for sweep i,j (flow in only)
**okb2(i,j)     second outer k boundary value of variable **          **floor
                for sweep i,j (flow in only)
**okb3(i,j)     third  outer k boundary value for emf's                  0.0
                for sweep i,j (flow in only)

       namelist / okb      /
     1              nokb    , dokb1   , dokb2   , v1okb1  , v1okb2
     2            , v2okb1  , v2okb2  , v3okb1  , v3okb2
*if -def,ISO
     3            , eokb1   , eokb2
*endif -ISO
*if def,GRAV
     4            , gpokb
*endif GRAV
*if def,MHD
     5            , b1okb1  , b1okb2  , b2okb1  , b2okb2  , b3okb1
     6            , b3okb2  , emf1okb1, emf1okb2, emf1okb3, emf2okb1
     7            , emf2okb2, emf2okb3, emf3okb1, emf3okb2, emf3okb3
*endif MHD
```

14. GRVCON – GRaVity CONtrol, read from subroutine NMLSTS

This namelist specifies the parameters which control the gravitational force from a point mass. In addition, the parameters to specify the type of boundary conditions for the self-gravity module are in this namelist.

```
parameter                    description                              default

g          gravitational constant                                     6.67e-8
ptmass     fixed central point mass                                   0.0
izero      i index of x1=0                                          is=3
jzero      j index of x2=0                                          js=3
kzero      k index of x3=0                                          ks=3
igpiib     0 => symmetric (Dirichlet) boundary conditions             0
igpoib     1 => non-symmetric boundary conditions (value              0
igpijb          calculated using multipole expansion)                 0
igpojb                                                                0
igpikb                                                                0
igpokb                                                                0

       namelist / grvcon   /
     1              g       , ptmass  , izero   , jzero   , kzero
     2            , igpiib  , igpoib  , igpijb  , igpojb  , igpikb
     3            , igpokb
```

15. EQOS – EQuation Of State control, read from subroutine NMLSTS

This namelist specifies the parameters which control the equation of state. Using all the defaults is recommended, unless a different adiabatic constant (gamma) is required. This version of ZEUS-3D does not permit multi-material calculations, and so the parameter mn should always be set to 1. Note that if an isothermal equation of state is desired, setting the EDITOR definition ISO in addition to setting niso = 1 will allow execution to take advantage of the reduced computations necessary for isothermal systems.

```
parameter                      description                              default

nmat          number of materials                                        1
gamma (mn)    ratio of specific heats                                   5/3
rgas  (mn)    gas constant                                               1.0
niso  (mn)    =0 => adiabatic  eos                                       0
              =1 => isothermal eos
ciso  (mn)    isothermal sound speed                                     1.0
rmetal(mn)    metallicity => cooling strength  M-MML                     0.0

       namelist / eqos     /
     1               nmat    , gamma   , rgas    , niso    , ciso
     2             , rmetal
```

16. GCON – Grid motion CONtrol, read from subroutine NMLSTS

This namelist sets the parameters for grid motion, should a partial tracking of the flow be required. This feature has only been partially tested in this release.

```
parameter                      description                              default

x1fac      x1 motion factor.                                             0.0
           < 0 gives "Lagrangian" tracking in x1 lines.
x2fac      x2 motion factor.                                             0.0
           < 0 gives "Lagrangian" tracking in x2 lines.
x3fac      x3 motion factor.                                             0.0
           < 0 gives "Lagrangian" tracking in x3 lines.
ia         i < ia, zone ratio is preserved in x1 lines                 is=3
ja         j < ja, zone ratio is preserved in x2 lines                 js=3
ka         k < ka, zone ratio is preserved in x3 lines                 ks=3
igcon      selects grid treatment:                                       0
           = 0  =>  separate motion
           = 1  =>  averaged motion
           = 2  =>  tracking x1, x2, and x3 boundary
           = 3  =>  averaged boundary tracking
           = 4  =>  input grid boundary speeds
                    vg1(io) = x1fac * central sound speed
                    vg2(jo) = x2fac * central sound speed
                    vg3(ko) = x3fac * central sound speed

       namelist / gcon     /
     1               x1fac   , x2fac   , x3fac   , ia      , ja
```

```
     2              , ka      , igcon
```

17. EXTCON – grid EXTension CONtrol, read from subroutine NMLSTS

This namelist controls the grid extension feature of the code. This is useful only for problems in which a shock separates quiescent material (which does not require updating) from material requiring computations. As the shock propagates across the grid, more zones are added to the computational domain until the entire domain has been included. Because quiescent zones are not being updated, a substantial savings in computation time could be realized. Use this feature with caution. Improper use can be disastrous.

```
parameter                        description                              default

istretch(1) .le.  0  => perform computations over entire i-domain          0
            .gt.  0  => i-index of first zone in initial i-domain
istretch(2)             i-index of last  zone in initial i-domain.         0
            .le.  (1) => istretch(2)=istretch(1)+istretch(3)-1
istretch(3) .le.  0  => 10                                                 0
istretch(4) .le.  0  => istretch(3)                                        0
```

is is decremented by istretch(3) and/or ie is incremented by istretch(4) whenever the quiescent density increases by 3% within 5 zones of the current domain boundary. Note that is will not be permitted to fall below ismn and ie will not be permitted to rise above iemx.

```
jstretch(1,2,3,4)       same as "istretch", but for the 2-direction.
kstretch(1,2,3,4)       same as "istretch", but for the 3-direction.

       namelist / extcon   /
     1               istretch, jstretch, kstretch
```

18. PLT1CON – PLoT (1-D) CONtrol, read from subroutine NMLSTS

This namelist controls the 1-D NCAR graphics. During a run, as many as nios 1-D slices may be specified for each variable plotted, where nios is a parameter set before compilation (currently, nios = 20). For every slice chosen, a metafile is created with a plot generated for each variable specified. Each 1-D slice continues across the entire data volume parallel to one of the axes of the computational grid. To uniquely specify the slice, two of iplt1, jplt1, and kplt1 must be set.

```
parameter                        description                              default

iplt1dir(nios)  axis parallel to slice.  0 => no plots                      0
                1, 2, 3 => 1-, 2-, 3-direction
iplt1   (nios)  i index of 1-D plot in 2- or 3-direction               (is+ie)/2
jplt1   (nios)  j index of 1-D plot in 3- or 1-direction               (js+je)/2
kplt1   (nios)  k index of 1-D plot in 1- or 2-direction               (ks+ke)/2
dtplt1          physical (problem) time interval between 1-D             0.0
```

```
                plot dumps.  0.0 => no plots.
plt1var (niov)  names of variables to be plotted (character*2).      'zz'
                Valid names are 'd ' (density), 'e ' (internal
                energy), 'se' (specific internal energy), 'p '
                (thermal pressure), 'pb' (magnetic pressure),
                'pt' (thermal plus magnetic pressure), 'v1',
                'v2', 'v3' (velocity components), 's1', 's2',
                's3' (momentum components), 'm ' (Mach number),
                'gp' (gravitational potential), 'b1', 'b2', 'b3'
                (magnetic field components), 'bd' (magnetic
                field / density), 'j1', 'j2', 'j3' (current
                density components).
nlplt1  (niov)  =0 => plot data                                      0
                =1 => plot log10 of data (positive definite
                      quantities only)
plt1min (niov)  minimum value to be plotted.                         0.0
plt1max (niov)  maximum value to be plotted.                         0.0
iplt1mm         =1 => compute "plt1min" and "plt1max" for plots      1
                =0 => use input "plt1min", "plt1max" for plots
                      If "plt1min" and "plt1max" are 0, compute
                      them as if "iplt1mm" were 1
corl            =1 => use open Circles, one per zone                 2
                =2 => use Line segments to connect zone values

      namelist / plt1con  /
     1              iplt1dir, iplt1   , jplt1   , kplt1   , dtplt1
     2            , plt1var , nlplt1  , plt1min , plt1max , iplt1mm
     3            , corl
```

19. PLT2CON – PLoT (2-D) CONtrol, read from subroutine NMLSTS

This namelist controls the 2-D NCAR graphics. During a run, as many as `nios` 2-D slices may be specified for each variable plotted. For every slice chosen, a metafile is created with a plot generated for each variable specified. The normal to each slice is parallel to one of the axes of the computational grid and is specified by `iplt2dir`. The extent of the slice is limited by `x1p2mn`, `x1p2mx`, *etc.*, while the index at the base of the normal to the slice is given by `lplt2`.

2-D graphics are in the form of contours (scalars and vector components normal to the image plane), vectors (poloidal vector components), or both for combined plots.

N.B. For restarted runs in which the computation is resumed on a larger or smaller grid, and where the default values for `x1p2mn`, `x1p2mx`, *etc.* were used in the initial run, it will be necessary to set `x1p2mn`, `x1p2mx`, *etc.* in the input deck for the restarted run to the extrema of the new grid if the plots are to extend to the bounds of the new grid. Otherwise, the plots will be bound by the old grid.

```
parameter                    description                          default

iplt2dir(nios)  direction of normal to image plane.                  0
                0 => no plots; 1, 2, 3 => 1-, 2-, 3-direction
lplt2   (nios)  level of 2-D plot (value of 1-, 2-, or            (is+ie)/2
```

```
                3-index)
dtplt2          physical (problem) time interval between 2-D          0.0
                plot dumps.  0.0 => no plots.
plt2var (niov)  names of variables to be plotted (character*2).      'zz'
                Valid names are 'd ', 'e ', 'se', 'p ', 'pb',
                'pt', 'vp' (poloidal velocity), 'vn' (normal
                velocity), 'vv' (divergence of velocity), 'sp'
                (poloidal momentum), 'sn' (normal momentum),
                'm ', 'gp', 'an' (normal vector potential), 'bp'
                (poloidal magnetic field), 'bn' (normal magnetic
                field), 'jp' (poloidal current density), 'jn'
                (normal current density), 'dv' (density plus
                poloidal velocity), 'ds' (density plus poloidal
                momentum), 'db' (density plus poloidal magnetic
                field), 'dj' (density plus poloidal current
                density).
nlplt2  (niov)  =0 => plot data                                       0
                =1 => plot log10 of data (positive definite
                      quantities only)
plt2min (niov)  minimum value to be contoured.                        0.0
plt2max (niov)  maximum value to be contoured.                        0.0
iplt2mm         =1 => compute "plt2min" and "plt2max" for plots       1
                =0 => use input "plt2min", "plt2max" for plots
                      If "plt2min" and "plt2max" are 0, compute
                      them as if "iplt2mm" were 1
vscale          scaling factor for vectors                            0.8
incvx           index increment in x-direction for vector plots       1
incvy           index increment in y-direction for vector plots       1
numcl           number of contour levels                             20
units           sets the angular units (character*2)                  'rd'
                'rd' => radians, 'pi' => units of pi radians
                'dg' => degrees
x1p2mn  (nios)  minimum x1 of plot window                    x1a(is)
x1p2mx  (nios)  maximum x1 of plot window                    x1a(ie+1)
x2p2mn  (nios)  minimum x2 of plot window                    x2a(js)
x2p2mx  (nios)  maximum x2 of plot window                    x2a(je+1)
x3p2mn  (nios)  minimum x3 of plot window                    x3a(ks)
x3p2mx  (nios)  maximum x3 of plot window                    x3a(ke+1)

      namelist / plt2con  /
     1              iplt2dir, lplt2   , dtplt2  , plt2var , nlplt2
     2            , plt2min , plt2max , iplt2mm , vscale  , incvx
     3            , incvy   , numcl   , units   , x1p2mn  , x1p2mx
     4            , x2p2mn  , x2p2mx  , x3p2mn  , x3p2mx
```

20. PIXCON – PIXel graphics CONtrol, read from subroutine NMLSTS

This namelist controls the pixel dumps. Pixel dumps are 2-D raster images of slices through the data volume, and are rebinned to a uniform, square Cartesian grid. During a run, as many as nios slices may be specified for each variable plotted. A single pixel dump is created for every variable and every slice specified. The extent of the pixel slice

can be limited by setting `x1pxmn`, `x1pxmx`, *etc.* The normal to the pixel slice is parallel to one of the axes of the computational grid and is specified by `ipixdir`. The index at the base of the normal is given by `lpix`.

Pixel dumps are designed to provide a format for generating smooth qualitative temporal animations of the flow variables. Aim for about 500 dumps for each animation. They may be written in either raw format (`rorhpix=1`, one byte per datum) or HDF (`rorhpix=2`, four bytes per datum).

Raw format files are small, and so numerous images may be generated with a relatively small amount of disc space. However, the low dynamic range of the images (256) dictates that the data be bracketed and perhaps even dumped logarithmically in order to render the salient features visible. The data may be bracketed automatically (`ipixmm=1`), in which case differences from one image to the next will be caused by both the evolution of the flow *and* the fluctuations of the extrema which are used to bracket the data. Alternatively, one may bracket the data manually (`ipixmm=0`) by setting values for `pixmin` and `pixmax`. This can be done by running the simulation until 10 to 20 pixel dumps have been generated for each variable with `ipixmm` set to 1. The extrema used to bracket the data are reported in the log file `zl`*nn*`id`, and these can be used to set the extrema `pixmin` and `pixmax`. Now run the job from the beginning with `ipixmm` set to 0. If a log dump is desired, some experimentation may have to be done in order to set the value of `nlpix` (the dynamic range) properly. However, the default value of 100 should be fine for most applications. Basically, the higher the absolute value for a positive (negative) `nlpix`, the more concentrated the colors will be at the low (high) end.

HDF files are four times as big, and thus may cause disc and storage problems. However, because these images are four bytes deep, bracketing and converting to log are not necessary. In fact, these files may be used quantitatively as well as qualitatively. For HDF, the parameters `ncpix`, `ipixmm`, `pixmin`, `pixmax`, and `nlpix` are all ignored.

Both Cartesian slices and polar slices are treated in the same way. With this release, Polar IMAGETOOL is no longer necessary. If a polar grid includes very small zones near the origin, it may be best to request two pixel slices for each slice to be visualized. One slice would include the entire grid and mimic the resolution near the mid-radial regions (*i.e.*, oversample the outer grid, but undersample the inner grid). The second slice would include only the inner radial regions and would mimic the resolution of the inner grid. In this way, the main advantage of polar IMAGETOOL and the previous format for polar pixel dumps may be recovered.

The parameters which set the dimensions of the arrays for the pixel plots (`nxpx, nypx`) are independent of the parameters which set the dimensions of the flow variables (`in,jn,kn`). Thus, in the case of a non-uniform grid, pixel dumps may be written with enough pixels to preserve the highest resolution on the grid.

N.B. For restarted runs in which the computation is resumed on a larger or smaller grid, and where the default values for `x1pxmn`, `x1pxmx`, *etc.* were used in the initial run, it will be necessary to set `x1pxmn`, `x1pxmx`, *etc.* in the input deck for the restarted run to the extrema of the new grid if the dumps are to extend to the bounds of the new grid. Otherwise, the dumps will be bound by the old grid.

parameter	description	default

```
ipixdir(nios)   direction of normal to image slice.                  0
                0 => no dumps; 1, 2, 3 => 1-, 2-, 3-direction
lpix   (nios)   level of 2-D pixel dump (value of 1-, 2-, or    (is+ie)/2
                3-index)
dtpix           problem time interval between pixel dumps            0.0
                0.0 => no pixel dumps
ncpix           number of color contour levels in image           253
iref            =0 => no reflection                                  0
                =1 => q reflected across x-axis on output,
                      generates twice the y-pixels requested
jref            =0 => no reflection                                  0
                =1 => q reflected across y-axis on output,
                      generates twice the x-pixels requested
npi    (nios)   number of x-pixels in image slice                   nxpx
npj    (nios)   number of y-pixels in image slice                   nypx
pixvar (niov)   names of variables to be plotted (character*2).     'zz'
                Valid names are 'a1', 'a2', 'a3' (vector potential
                components), 'b1', 'b2', 'b3', 'd ', 'e ', 'gp',
                'j1', 'j2', 'j3', 'm ', 'p ', 'pb', 'pt', 's1',
                's2', 's3', 'se', 'v1', 'v2', 'v3', 'vv'
nlpix  (niov)   =0 => store data                                     0
                >0 => store log10(data), concentrating colors at
                      low end.  Dynamic range = nlpix, 1 => 100.
                <0 => store log10(data), concentrating colors at
                      high end.  Dynamic range =-nlpix, -1 => -100.
pixmin (niov)   value of data to be assigned the minimum color.    0.0
pixmax (niov)   value of data to be assigned the maximum color.    0.0
ipixmm          =1 => compute "pixmin" and "pixmax" for images       1
                =0 => use input "pixmin", "pixmax" for images
                      If "pixmin" and "pixmax" are 0, compute
                      them as if "ipixmm" were 1
rorhpix         =1 => raw format used for dumps                      1
                =2 => HDF used for dumps (in which case, "nlpix",
                      "pixmin", and "pixmax" are ignored)
units           sets the angular units (character*2)                 'rd'
                'rd' => radians, 'pi' => units of pi radians
                'dg' => degrees
x1pxmn (nios)   minimum x1 for pixel image                      x1a(is)
x1pxmx (nios)   maximum x1 for pixel image                      x1a(ie+1)
x2pxmn (nios)   minimum x2 for pixel image                      x2a(js)
x2pxmx (nios)   maximum x2 for pixel image                      x2a(je+1)
x3pxmn (nios)   minimum x3 for pixel image                      x3a(ks)
x3pxmx (nios)   maximum x3 for pixel image                      x3a(ke+1)
```

```
      namelist / pixcon   /
     1              ipixdir , lpix    , dtpix   , ncpix   , iref
     2            , jref    , npi     , npj     , pixvar  , nlpix
     3            , pixmin  , pixmax  , ipixmm  , rorhpix , units
     4            , x1pxmn  , x1pxmx  , x2pxmn  , x2pxmx  , x3pxmn
     5            , x3pxmx
```

21. VOXCON – VOXel graphics CONtrol, read from subroutine NMLSTS

This namelist controls the voxel dumps of the 3-D data volume. These are the 3-D analogues of the 2-D pixel dumps, and are snapshots of the entire data volume. See comments in namelist `pixcon` above for discussion on raw format *vs.* HDF, bracketing, and dumping files logarithmically.

Voxel dumps are currently available for Cartesian (`XYZ`) and cylindrical (`ZRP`) geometries only. The dimensions of the voxel dumps are limited by the parameters `in`, `jn`, and `kn`. In particular, the voxel dump may be no larger than `in-1` $\times$ `2*jn-1` $\times$ `2*kn-1`. For a uniform Cartesian grid, there is no reason to specify a voxel dump larger than the flow variable array. However, for non-uniform gridding in either or both of the 2- and 3-directions in `XYZ` coordinates, or in `ZRP` coordinates in general, the factor of 2 in both of these dimensions will allow the voxel dumps to represent better the regions in the computational grid with the highest resolution. 250 voxel dumps with four million voxels (from a one million zone computation) will require 1 Gbyte of disc space. Thus, temporal sequences of 3D voxel dumps are possible, but only in a limited fashion.

N.B. For restarted runs in which the computation is resumed on a larger or smaller grid, and where the default values for `x1vxmn`, `x1vxmx`, *etc.* were used in the initial run, it will be necessary to set `x1vxmn`, `x1vxmx`, *etc.* in the input deck for the restarted run to the extrema of the new grid if the dumps are to extend to the bounds of the new grid. Otherwise, the dumps will be bound by the old grid.

parameter	description	default
dtvox	problem time interval between voxel dumps 0.0 => no voxel dumps.	0.0
ncvox	number of color contour levels in image	253
nvi	number of voxels in 1-direction (.le. in-1) =0 => in-1	0
nvj	number of voxels in 2-direction (.le. 2*jn-1) =0 => increment in 2-dir. same as 1-dir.	0
nvk	number of voxels in 3-direction (.le. 2*kn-1) =0 => increment in 3-dir. same as 1-dir.	0
voxvar(niov)	names of variables to be plotted (character*2). Valid names are 'b1', 'b2', 'b3', 'd ', 'e ', 'gp', 'j ', 'j1', 'j2', 'j3', 'm ', 'p ', 'pb', 'pt', 's1', 's2', 's3', 'se', 'v ', 'v1', 'v2', 'v3', 'vv'	'zz'
nlvox (niov)	=0 => store data >0 => store log10(data), concentrating colors at low end. Dynamic range = nlvox, 1 => 100. <0 => store log10(data), concentrating colors at high end. Dynamic range =-nlvox, -1 => -100.	0
voxmin(niov)	value of data to be assigned the minimum color.	0.0
voxmax(niov)	value of data to be assigned the maximum color.	0.0
ivoxmm	=1 => compute "voxmin" and "voxmax" for images =0 => use input "voxmin", "voxmax" for images If "voxmin" and "voxmax" are 0, compute them as if "ivoxmm" were 1	1
rorhvox	=1 => raw format used for dumps =2 => HDF used for dumps (in which case, "nlvox", "voxmin", and "voxmax" are ignored)	1

```
units           sets the angular units (character*2)                'rd'
                'rd' => radians, 'pi' => units of pi radians
                'dg' => degrees
x1vxmn          minimum x1 for voxel image                      x1a(is)
x1vxmx          maximum x1 for voxel image                      x1a(ie+1)
x2vxmn          minimum x2 for voxel image                      x2a(js)
x2vxmx          maximum x2 for voxel image                      x2a(je+1)
x3vxmn          minimum x3 for voxel image                      x3a(ks)
x3vxmx          maximum x3 for voxel image                      x3a(ke+1)

       namelist / voxcon   /
     1              dtvox   , ncvox   , nvi     , nvj     , nvk
     2            , voxvar  , nlvox   , voxmin  , voxmax  , ivoxmm
     3            , rorhvox , units   , x1vxmn  , x1vxmx  , x2vxmn
     4            , x2vxmx  , x3vxmn  , x3vxmx
```

22. USRCON – USeR dump CONtrol, read from subroutine NMLSTS

This namelist is reserved for a user-supplied I/O subroutine aliased to USERDUMP (see Appendix 1).

```
parameter                       description                         default

dtusr      physical (problem) time interval between user dumps.      0.0
           0.0 => no user dumps

       namelist / usrcon   /
     1              dtusr
```

23. HDFCON – HDF dump CONtrol, read from subroutine NMLSTS

This namelist controls the HDF dumps. HDF is a format for data files developed at NCSA, and is becoming widely used throughout the world. HDF dumps are necessary for most of the graphics packages available. These files are 4 bytes deep, and contain the grid coordinates along with other useful information about the data.

```
parameter                       description                         default

dthdf           physical (problem) time interval between hdf         0.0
                dumps.  0.0 => no hdf dumps
hdfvar(niov)    names of variables to be dumped (character*2).       'zz'
                Valid names are 'to' ("total" dump => v1, v2,
                v3, b1, b2, b3, d, and e all in the same file),
                'b1', 'b2', 'b3', 'd ', 'e ', 'gp', 'j1', 'j2',
                'j3', 'm ', 'p ', 'pb', 'pt', 's1', 's2', 's3',
                'se', 'v1', 'v2', 'v3'

       namelist / hdfcon   /
     1              dthdf   , hdfvar
```

24. TSLCON – Time SLice (history) dump CONtrol, read from subroutine NMLSTS

This namelist controls the time slice data dumps. Various scalars, such as total mass, angular momenta, energy, extrema of variables, *etc.* are periodically written to an ascii file and/or a metafile (NCAR graphics).

parameter	description	default
dttsl	physical (problem) time interval between time slice ascii dumps. 0.0 => no ascii time slices	0.0
dttslp	physical (problem) time interval between time slice plot dumps. 0.0 => no metacode time slices	0.0
tslpmn	problem time for beginning of plot	0.0
tslpmx	problem time for end of plot (0.0 => maximum time)	0.0

```
      namelist / tslcon   /
    1               dttsl   , dttslp  , tslpmn  , tslpmx
```

25. DISCON – DISplay dump CONtrol, read from subroutine NMLSTS

This namelist controls the display dumps of 2-D slices. During a run, as many as nios slices may be specified for each variable displayed. All display dumps generated during a run are dumped to the same ascii data file. The extent of the display slice can be limited by setting idismn, idismx, *etc.* The normal to the display slice is parallel to one of the axes of the computational grid and is specified by idisdir. The index at the base of the normal is given by ldis.

The display format allows the user to view a small portion of the data quantitatively in a matrix format. The maximum amount of data that can be visualized at once from each specified variable and slice is 38 by 38. The data are scaled and converted to integers with a dynamic range anywhere from 100 to 10^6, depending on the amount of data being displayed. The data are arranged in a 2-D matrix and labelled with the grid indices and the scaling factor used to scale the data. (The functionality is similar to that of the task PRTIM in AIPS.)

N.B. For restarted runs in which the computation is resumed on a larger or smaller grid, and where the default values for idismn, idismx, *etc.* were used in the initial run, it will be necessary to set idismn, idismx, *etc.* in the input deck for the restarted run to the extrema of the new grid if the dumps are to extend to the bounds of the new grid. Otherwise, the dumps will be bound by the old grid.

parameter	description	default
idisdir(nios)	direction of normal to display slice: 0 => no dumps; 1, 2, 3 => 1-, 2-, 3-direction	0
ldis (nios)	level of 2-D display (value of 1-, 2-, or 3-index)	(is+ie)/2
dtdis	physical (problem) time interval between display dumps. 0.0 => no display dumps.	0.0
disvar (niov)	names of variables to be displayed (character*2).	'zz'

```
                Valid names are 'd ', 'e ', 'se', 'p ', 'pb',
                'pt', 'v1', 'v2', 'v3', 'vv', 's1', 's2', 's3',
                'm ', 'gp', 'b1', 'b2', 'b3', 'j1', 'j2', 'j3'
idismn (nios)   bottom i-index of display window                    is
idismx (nios)   top    i-index of display window                    ie
jdismn (nios)   bottom j-index of display window                    js
jdismx (nios)   top    j-index of display window                    je
kdismn (nios)   bottom k-index of display window                    ks
kdismx (nios)   top    k-index of display window                    ke

      namelist / discon   /
     1              idisdir , ldis    , dtdis   , disvar  , idismn
     2             , idismx  , jdismn  , jdismx  , kdismn  , kdismx
```

26. RADCON – RADio dump CONtrol, read from subroutine NMLSTS

This namelist controls the RADIO dumps, which are 2-D pixel dumps of quantities integrated along the lines of sight through the data volume at arbitrary viewing angles (`theta` and `phi`). The volume integrated can be limited by setting `x1rdmn`, `x1rdmx`, *etc.* RADIO dumps are currently available for Cartesian (`XYZ`) and cylindrical (`ZRP`) geometries. See discussion in namelist `pixcon` regarding raw format *vs.* HDF, bracketing images, and dumping images logarithmically.

There are two types of integrated quantities: flow variables and emissivities. Many of the parameters listed below are for controlling the latter. For example, the Stokes parameters once integrated can be convolved with a beam, polarization vectors may be plotted directly (rather than raster images), polarization vectors may be superposed on total intensity raster images, and so on.

The "masks" (`*lower`, `*upper`, `btsw`, and `dsw`) are useful in limiting which portion of the grid is included in the integration of the non-emissivity scalars. For example, if there is a contact discontinuity (CD) enclosing the region of interest, then there will be a jump in the density (`d`) along this interface. Thus, if `d`, for example, jumps from about 0.1 to about 1.0 across the CD, setting `dsw=1.0`, and `dupper=0.5` would allow only the low density region (be it interior or exterior to the CD) to contribute to the line-of-sight integration. Alternatively, if the magnetic field is found only in the material of interest, setting `btsw=1.0` would allow only material with magnetic field to be included in the integration. Finally, the variables `*lower` and `*upper` allow each variable to be masked by its own distribution. These can be set in addition to the density and/or magnetic field masks (`dsw`, `btsw`). For example, if only the compressive portions of the flow are to be integrated, then setting `xupper=0.0` will mean that only negative values of $\nabla \cdot \mathbf{v}$ will be included in the integration. All values excluded by the various masks will be given zero weight. In all cases, the default is no mask.

Reversing the palette (`nlrad<0`) is useful for raster images in which `radmin<0` and `radmax<0` (*e.g.*, negative velocity divergences). In these cases, it may be desirable to have the "maximum" color correspond to the minimum pixel value (which has the greatest absolute value).

Note that the parameters which set the dimensions of the arrays for the RADIO

pixel plots (nxrd,nyrd) are independent of the parameters which set the dimensions of the flow variables (in,jn,kn) and of the regular pixel slices (nxpx,nypx).

N.B. For restarted runs in which the computation is resumed on a larger or smaller grid, and where the default values for x1rdmn, x1rdmx, *etc.* were used in the initial run, it will be necessary to set x1rdmn, x1rdmx, *etc.* in the input deck for the restarted run to the extrema of the new grid if the dumps are to extend to the bounds of the new grid. Otherwise, the dumps will be bound by the old grid.

parameter	description	default
dtrad	problem time interval between RADIO dumps 0.0 => no RADIO dumps.	0.0
thetamin	minimum angle between x1-axis and plane of sky	0.0
thetamax	maximum angle between x1-axis and plane of sky	0.0
dtheta	desired increment in "theta" between successive dumps	0.0
ntheta	number of values for "theta" between specified limits (overrides choice for "dtheta")	1
phimin	minimum azimuthal angle for lines of sight.	0.0
phimax	maximum azimuthal angle for lines of sight.	0.0
dphi	desired increment in "phi" between successive dumps	0.0
nphi	number of values for "phi" between specified limits (overrides choice for "dphi")	1
itype	0 => emissivities are not computed. 1 => Smith et al. emissivity (p**2) 2 => CNB emissivity (function of d, p, B)	2
alpha	spectral index (itype=2 only)	0.5
freq	frequency of RADIO "observation" (Hz).	5.0e9
ncrad	number of color contour levels in images	253
radvar (niov)	names of variables to be plotted (character*2). Currently, valid names are , A (pol'n position angle), AV (pol'n position angle with pol'n vectors superposed), F (P/I), FV (P/I with pol'n vectors superposed), I (total intensity), IV (total intensity with pol'n vectors superposed), P (pol'd intensity), PV (pol'd intensity with pol'n vectors superposed), V (pol'n vectors, black on white), VN (pol'n vectors, white on black), D (density), E (internal energy), SE (specific internal energy), PB (magnetic pressure), SH (velocity shear), VV (velocity divergence).	'zz'
*lower	variable * is integrated along los provided it is greater than "*lower", where * is any one of d, e, se, pb, sh, and vv.	-huge
*upper	variable * is integrated along los provided it is less than "*upper".	huge
dsw	1.0 => "dlower" and "dupper" determine integration limits for all variables., 0.0 => not	0.0
btsw	1.0 => B-field extent determines integration limits for all variables.	0.0

```
                0.0 => not
nlrad  (niov)   =0 => store data                                        0
                >0 => store log10(data), concentrating colors at
                      low end.  Dynamic range = nlrad, 1 => 100.
                <0 => store log10(data), concentrating colors at
                      high end.  Dynamic range =-nlrad, -1 => -100.
radmin (niov)   value of data to be assigned the minimum color.      0.0
radmax (niov)   value of data to be assigned the maximum color.      0.0
iradmm          =1 => compute "radmin" and "radmax" for images        1
                =0 => use input "radmin", "radmax" for images
                      If "radmin" and "radmax" are 0, compute
                      them as if "iradmm" were 1
rorhrad         =1 => raw format used for dumps                       1
                =2 => HDF used for dumps (in which case, "nlrad",
                      "radmin", and "radmax" are ignored)
icnvlv          0 => do not apply convolution                         0
                1 => apply convolution to Stokes parameters.
bmajor          major axis of convolving beam.                        1.0
bminor          minor axis of convolving beam.                        1.0
bpa             beam position angle (radians) measured counter-       0.0
                clockwise between major axis and +x axis.
cpb             "cells" per beam.                                     5.0
eorb            1 => E-vectors                                        2
                2 => B-vectors
porf            1 => vector length proportional to poli               2
                2 => vector length proportional to fpol
bworb           1 => black and white pixel vectors                    1
                2 => black pixel vectors only
vlmin           vectors with length < vlmin*(max vector) not          0.001
                plotted.
icut            vectors are not plotted if toti < icut*max(toti)      0.001
pcut            vectors are not plotted if poli < pcut*max(poli)      0.001
pscale          scaling factor for polarization vectors               0.8
incpx           plot every incpx(th) vector in x-direction            1
incpy           plot every incpy(th) vector in y-direction            1
units           sets the angular units (character*2)                  'rd'
                'rd' => radians, 'pi' => units of pi radians
                'dg' => degrees
x1rdmn          minimum x1 for RADIO integration                x1a(is)
x1rdmx          maximum x1 for RADIO integration                x1a(ie+1)
x2rdmn          minimum x2 for RADIO integration                x2a(js)
x2rdmx          maximum x2 for RADIO integration                x2a(je+1)
x3rdmn          minimum x3 for RADIO integration                x3a(ks)
x3rdmx          maximum x3 for RADIO integration                x3a(ke+1)

      namelist / radcon   /
     1              dtrad   , thetamin, thetamax, ntheta  , dtheta
     3            , phimin  , phimax  , nphi    , dphi    , itype
     4            , alpha   , freq    , ncrad   , radvar  , dlower
     5            , elower  , tlower  , olower  , slower  , xlower
     6            , dupper  , eupper  , tupper  , oupper  , supper
     7            , xupper  , dsw     , btsw    , nlrad   , radmin
     8            , radmax  , iradmm  , rorhrad , icnvlv  , bmajor
```

```
9              , bminor  , bpa     , cpb     , eorb    , porf
1              , bworb   , vlmin   , icut    , pcut    , pscale
2              , incpx   , incpy   , units   , x1rdmn  , x1rdmx
3              , x2rdmn  , x2rdmx  , x3rdmn  , x3rdmx
```

27. PGEN – Problem GENerator, read from subroutine aliased to PROBLEM

This namelist is reserved for the problem generator, which sets the flow variables to the desired initial conditions. Thus the parameters which appear in this namelist depend on which problem is being studied. The desired problem is specified by setting the EDITOR alias PROBLEM in the file zeus32.mac to the name of the problem generating subroutine. This subroutine should initialize the active zones of all field variables and then call the subroutines bndyflgs, bndyall, and bndymag to set all the boundary values. See the problem generator template in Subsection 6.1.

Below is a description of some of the problem generators which are already part of the main source code zeus32.

---

1. ADVECT: Sets up an advection test in the 1-, 2-, or 3-direction. Periodic boundary conditions are used in XYZ geometry. In ZRP geometry, reflecting boundary conditions are used at the ijb and either flow in or flow out boundary conditions are used at the ojb, depending on the sign of v0. In RTP geometry, reflecting boundary conditions are used at the iib and either flow in or flow out boundary conditions are used at the oib, depending on the sign of v0.

```
parameter                    description                          default

idirect          1 => 1-direction                    ie biggest => 1
                 2 => 2-direction                    je biggest => 2
                 3 => 3-direction                    ke biggest => 3
advvar(nvmx)     variable names to be initialized.  Valid            'zz'
                 names are  'd ', 'e ', 'v1', 'v2', 'v3',
                 'b1', 'b2', and 'b3'.  NOTE: v* and b* can
                 not be advected in the *-direction, where
                 * = 1, 2, or 3.
ishp  (nvmx)     integer flag describing the shape of the pulse.     1
                 1 => square, 2 => triangular, 3 => ramps,
                 4 => Gaussian, 5 => co-sinusoidal, 6 => coordinate
amp   (nvmx)     amplitude of pulse                                  1.0
floor (nvmx)     amplitude outside of pulse                          0.1
m0    (nvmx)     grid zone at which pulse center is located        nx?z/2
n0    (nvmx)     ishp = 1,2,3 => initial width of pulse in zones   nx?z/5
                 ishp = 4     => number of zones per FWHM
                 ishp = 5     => number of zones per wavelength
                 ishp = 6     => irrelevant
d0               background density (if density pulse not advected)  1.0
v0               velocity of advection.                              1.0

For XYZ geometry  v1 = v0       v2 = v0                v3 = v0
```

```
For ZRP geometry  v1 = v0       v2 = v0 * x2            v3 = v0
For RTP geometry  v1 = v0 * x1  v2 = v0 * x1 * sin(x2)  v3 = v0

      namelist / pgen     /
     1              idirect , advvar  , ishp    , amp     , floor
     2               , m0     , n0     , d0      , v0
```

---

2. BLAST: Sets up an overpressured region centered at the specified point on the grid (x10,x20,x30) with the specified radius (r). The pressure and density ratios of the compressed region relative to the ambient are prat and drat respectively.

parameter	description	default
r	initial radius of overpressured region	1.0
x10,x20,x30	coordinates of center of overpressured region.	3*0.0
drat	ratio of density across blast front	1.0
prat	ratio of pressure across blast front	1.0e6
d0	density in ambient medium	1.0
p0	pressure in ambient medium	0.6
b10	1-magnetic field in ambient medium	0.0
b20	2-magnetic field in ambient medium	0.0
b30	3-magnetic field in ambient medium	0.0
m,drs,dr	parameters for specifying a sphere whose surface	0
	is sinusoidally perturbed (spherical coordinates	0.0
	only). For an unperturbed sphere, set all values	0.0
	to zero.	

```
      namelist / pgen     /
     1              r       , x10     , x20     , x30     , drat
     2               , prat    , d0      , p0      , b10     , b20
     3               , b30     , m       , drs     , drc
```

---

3. HOMOL: Sets up a homologous contraction or expansion test problem in one of the three directions.

parameter	description	default
idirect	1 => 1-direction	1
	2 => 2-direction	
	3 => 3-direction	
v0	velocity of contraction or expansion.	1.0
nn	exponent of velocity field. v =, v0 * r**nn	1
d0	density at x1a(i) = 1. d = d0 / r**nn	1.0
e0	internal energy density at x1a(i) = 1.0	1.0
	e = e0 / r**nn	

```
      namelist / pgen     /
     1              idirect , nn      , v0      , d0      , e0
```

4. SHKSET: Sets up a Sod shock tube in one of the three directions.

```
parameter                    description                          default

idirect    1 => 1-direction                          ie biggest => 1
           2 => 2-direction                          je biggest => 2
           3 => 3-direction                          ke biggest => 3
n0         number of zones to be initialized.  Namelist       nx1z
           cards are read from logical unit ioin until
           ie-is+1 (or je-js+1, or ke-ks+1) zones are
           initialized.
d0         input density                                        tiny
e0         input specific internal energy ( = e/d )             tiny
v10        input velocity in 1 direction                        0.0
v20        input velocity in 2 direction                        0.0
v30        input velocity in 3 direction                        0.0
b10        input magnetic field in 1 direction                  0.0
b20        input magnetic field in 2 direction                  0.0
b30        input magnetic field in 3 direction                  0.0

      namelist / pgen     /
     1               idirect , n0      , d0      , e0      , v10
     2             , v20     , v30     , b10     , b20     , b30
```

---

5. ZPINCH: Sets up a Bennett pinch equilibrium (Jackson, second edition, section 10.5, p. 479) in ZRP geometry.

```
parameter                    description                          default

d0         input density                                         [none]
e0         input internal energy (proportional to pressure)      [none]

      namelist / pgen     /
     1               d0      , e0
```

---

6. MHDROT: Sets up the aligned rotator problem which tests the MoC algorithm in ZRP coordinates.

```
parameter                    description                          default

rc         radius of the cylinder                                  1.0
omegac     initial constant angular velocity of the cylinder       1.0e-6
dc         density of the cylinder                                 1.0
da         density of the ambient medium                           1.0
ec         internal energy of the cylinder                         1.0
ea         internal energy of the ambient medium                   1.0
bi         axial magnetic field                                    1.0
```

```
      namelist / pgen     /
     1               rc      , omegac   , dc      , da      , ec
     2             , ea      , bi
```

---

7. JETINIT: Sets up the initial ambient medium and the jet orifice at the inner i boundary. An entire "equilibrium" jet may be established across the entire grid if desired. The orifice is centered on the origin (which must lie on the inner i-boundary) and is rounded as well as is possible in Cartesian coordinates. The inflow boundary conditions are set at all grid points inside the orifice. This overwrites any parameters that may have been set for these zones by the namelist `iib`. Thus, should a jet run be restarted, it is necessary to prevent the namelist `iib` from reseting all the inner i boundary parameters in the orifice zones. The jet orifice can be wiggled helically (to break the pseudo-azimuthal symmetry) by setting `iwiggle` to 1 and by setting the EDITOR alias `BNDYUPDATE` to `wiggle`. For a circular helical perturbation, set `thetaj=thetak`. For an elliptical helical perturbation, set `thetaj`$\neq$`thetak`. For a sinusoidal perturbation, set one of `thetaj` or `thetak` to zero. To generate a magnetic field at the jet orifice, set `BNDYUPDATE` to `bgen`. If both `wiggle` and `bgen` are required, set `BNDYUPDATE` to `jetbndy`, which calls both subroutines in succession.

   If a non-uniform atmosphere is required, the user may use the EDITOR alias `ATMOSPHERE` to include a user-supplied subroutine describing the desired atmosphere.

```
parameter                        description                                  default

Input parameters for the jet:

hydrodynamical variables:
rjet       radius of jet (in coordinate units)                                 1.0
eta        ratio between jet density and ambient medium density               0.1
           effective at the inflow boundary
kappa      ratio between jet pressure and ambient medium pressure             1.0
           effective at the inflow boundary
mjet       Mach number of the jet with respect to its own sound               6.0
           speed at the jet inlet
ojet      initial angular velocity of jet about axis (solid body              0.0
          rotation
pore      =1 => propagating jet                                               1
          =2 => equilibrium jet (Phil Hardee's stability runs)
b1jet     initial magnetic field in 1-direction (pore=2 only)                 0.0

toroidal magnetic field generator:
betator    >0 => minimum plasma beta of toroidal magnetic field              huge
           <0 => average plasma beta of toroidal magnetic field
rpeak      radial distance from the jet axis to the peak of the               0.5
           toroidal magnetic field profile
rsigma     FWHM of the toroidal magnetic field profile in radial              0.5
           direction

poloidal magnetic field generator:
```

```
betapol    >0 => minimum plasma beta of poloidal magnetic field      huge
           <0 => average plasma beta of poloidal magnetic field
ibpol      =1 => semi-infinite flux loop of axial magnetic field,    1
                 ramped up using "zsigma".
           =2 => flux loops of axial magnetic field
zsigma     HWHM of the magnetic field profile in the axial           0.0
           direction (to "ramp up" the magnetic field)
rnode      radial distance between flux loop nodes                   1.0
znode      axial distance between flux loop nodes                    1.0

random magnetic field generator:
betaran    effective plasma beta of the random magnetic field        huge
           component (0 => huge).
           .lt.  huge => toroidal and poloidal generators turned off.

Input parameters for the ambient medium:

damb       density                                                   1.0
eamb       specific internal energy (= e/d).  Default of 0.9         0.9
           implies an external sound speed of 1.0.
v1amb      velocity in 1 direction                                   0.0
v2amb      velocity in 2 direction                                   0.0
v3amb      velocity in 3 direction                                   0.0
b1amb      magnetic field in 1 direction                             0.0
b2amb      magnetic field in 2 direction                             0.0
b3amb      magnetic field in 3 direction                             0.0

Input parameters for wiggling the jet at the orifice:

iwiggle    =0 => do not wiggle jet orifice                           0
           =1 => wiggle jet orifice
omega      frequency of sinusoidal perturbation (in units of         0.0
           external sound speed divided by the jet radius)
thetaj     maximum ratio of v2 to v1 at inlet                        0.02
thetak     maximum ratio of v3 to v1 at inlet                        0.02
rmode      resonant frequency mode                                   1.0
```

Note that if `omega` is specified, it is not necessary to specify `rmode`.

```
      namelist / pgen     /
     1             , rjet    , eta     , kappa   , mjet    , ojet
     1             , pore    , b1jet   , betator , rpeak   , rsigma
     1             , betapol , ibpol   , zsigma  , rnode   , znode
     1             , betaran , damb    , eamb    , v1amb   , v2amb
     3             , v3amb   , b1amb   , b2amb   , b3amb   , iwiggle
     4             , omega   , thetaj  , thetak  , rmode
```

APPENDIX 3: THE ZEUS-3D VARIABLES

This Appendix contains a glossary of the variables used in `zeus32`, and is meant to aid the user in writing subroutines and making changes to the source code itself. It is by no means complete, but should contain the variables needed for most purposes. All these variables are declared in the common deck `rouse`. Thus, adding the EDITOR command `*call rouse` before the local declarations makes all these variables accessible from within the subroutine.

The user should be aware of the index convention used. A 3-D array, such as the density, is denoted `d(i,j,k)`, where `i` is the index for the `x1` coordinate, `j` is the index for the `x2` coordinate, and `k` is the index for the `x3` coordinate. The coordinates `x1`, `x2`, and `x3` are intentionally generic, since an attempt has been made to write the code in a covariant fashion. In Cartesian, cylindrical, and spherical polar coordinates, `(x1,x2,x3)` corresponds to `(x,y,z)`, `(z,r,`ϕ`)`, and `(`ρ`,`θ`,`ϕ`)` respectively. In FORTRAN, the index which changes the fastest is the first one. Thus, in triple do-loops which manipulate the 3-D arrays, it is best to have the outer loop run on `k`, the middle loop run on `j`, and the inner loop run on `i`. If one of the directions is divided into more zones than the other two, then it is best that this direction be the 1-direction (with index `i`) since it is the inner loop which vectorizes. In Cartesian coordinates, this can always be arranged. The indices strictly follow a right hand rule. Thus, the array `nijb(k,i)` is a 2-D array which has `k` as its first index and `i` as its second. In the list below, arrays are given with their indexing to remind the user of the ZEUS-3D convention.

The user should also be aware of the gridding. The computational domain is divided into `in` by `jn` by `kn` zones. In each direction, five of these zones are "ghost" or "boundary" zones, while the remaining zones are "active" zones where the equations of MHD are solved. In Cartesian geometry, these zones are rectangular boxes. In general, the gridding need not be uniform, so the ratio of the dimensions of each zone need not be constant across the grid. There are eight locations one can associate uniquely with each zone. Each of these locations can be tagged with the indices `(i,j,k)`. These locations are: the center of each box, the center of three of the six faces, the center of three of the twelve edges, and one of the eight corners. In ZEUS-3D, there are two grids which are referred to as the half-grid (or the a-grid) and the full grid (or the b-grid). By convention, the `(i,j,k)`th point on the a-grid is half a grid spacing closer in each dimension to the origin that the `(i,j,k)`th point on the b-grid. Points on the b-grid `(x1b(i),x2b(j),x3b(k))` correspond to zone centers while points on the a-grid `(x1a(i),x2a(j),x3a(k))` correspond to zone corners. Edges and faces have mixed grid coordinates. The center of the 1-face has coordinates `(x1a(i),x2b(j),x3b(k))`, the center of the 2-face has coordinates `(x1b(i),x2a(j),x3b(k))`, and the center of the 3-face has coordinates `(x1b(i),x2b(j),x3a(k))`. The center of the 1-edge has coordinates `(x1b(i),x2a(j),x3a(k))`, the center of the 2-edge has coordinates `(x1a(i),x2b(j),x3a(k))`, and the center of the 3-edge has coordinates `(x1a(i),x2a(j),x3b(k))`.

For various reasons, it is necessary to "stagger" the grid. That is to say, not all variables are located at the same place. Scalars (density and internal energy) are zone-centered quantities while the components of the flow vectors (velocity and magnetic field)

are face-centered quantities penetrating the face upon which they are centered. Vectors derived from vector quantities such as the current density ($\nabla \times \mathbf{B}$) and the emf ($\mathbf{v} \times \mathbf{B}$) have edge-centered components parallel to the edges while scalars derived from vector quantities such as $\nabla \cdot \mathbf{v}$ are zone-centered. Thus, the two grids play equally important roles, and the user needs to be careful about which grid should be used and where the variables are located while making any changes to the code.

A3.1 GRID VARIABLES

Limits for do-loops:

Variable	Location	Description
`is, ie`		beginning and ending `i`-index for active zones
`js, je`		beginning and ending `j`-index for active zones
`ks, ke`		beginning and ending `k`-index for active zones

Corresponding to each variable (`is`, `ie`, *etc.*) are the limiting variables (`ismn`, `iemx`, *etc.*) which indicate the extreme values possible for the do-loop indices should the grid extending option be used (see the description of the namelist `extcon` in Appendix 2). In addition, the variables `ism2`, `ism1`, `isp1`, `isp2`, and `isp3` exist which are set to `is-2`, `is-1`, `is+1`, `is+2`, and `is+3` respectively. If the computation is symmetric in the `i`-direction, `ism2`, `ism1`, `isp1`, `isp2`, and `isp3` are simply set to `is`. Similar variables exist for `ie`, `js`, `je`, `ks`, and `ke`.

The b-grid:

Variable	Location	Description
`x1b(i)`	zone-center	`x1`-coordinate in grid units
`x2b(j)`	zone-center	`x2`-coordinate in grid units (radians in spherical polar coordinates)
`x3b(k)`	zone-center	`x3`-coordinate in grid units (radians in both cylindrical and spherical polar coordinates)
`dx1b(i)`	1-face	`x1b(i) - x1b(i-1)`
`dx2b(j)`	2-face	`x2b(j) - x2b(j-1)`
`dx3b(k)`	3-face	`x3b(k) - x3b(k-1)`

In order to make the grid covariant, metric factors have been introduced which carry all the dependence of the geometry. In general, the metric appears in the expression for a differential in volume:

$$dV = g_1 dx_1 \, g_2 dx_2 \, g_3 dx_3$$

In Cartesian coordinates, $g_1 = g_2 = g_3 = 1$. In cylindrical coordinates, $g_1 = g_2 = 1$, $g_3 = x_2$. In spherical polar coordinates, $g_1 = 1$, $g_2 = x_1$, $g_3 = x_1 \sin x_2$. Note that if one is limited to `XYZ`, `ZRP`, and `RTP` coordinates, there is no need for g_1 and g_3 can be split into two variables, one dependent just on x_1, the other just on x_2. In this way, g_3 can be represented by two 1-D arrays rather than one 2-D array. Thus, three 1-D metric factors

are used in ZEUS-3D.

`g2b(i)`	zone-center	`= 1` for Cartesian and cylindrical coordinates, `= x1b(i)` for spherical polar coordinates
`g31b(i)`	zone-center	`= g2b(i)`
`g32b(j)`	zone-center	`= 1` for Cartesian coordinates, `= x2b(j)` for cylindrical coordinates, `= sin(x2b(j))` for spherical polar coordinates

The a-grid:

Variable	Location	Description
`x1a(i)`	zone-corner	`x1`-coordinate in grid units
`x2a(j)`	zone-corner	`x2`-coordinate in grid units
`x3a(k)`	zone-corner	`x3`-coordinate in grid units
`dx1a(i)`	1-edge	`x1a(i+1) - x1a(i)`
`dx2a(j)`	2-edge	`x2a(j+1) - x2a(j)`
`dx3a(k)`	3-edge	`x3a(k+1) - x3a(k)`
`g2a(i)`	zone-corner	`= 1` for Cartesian and cylindrical coordinates, `= x1a(i)` for spherical polar coordinates
`g31a(i)`	zone-corner	`= g2a(i)`
`g32a(j)`	zone-corner	`= 1` for Cartesian coordinates, `= x2a(j)` for cylindrical coordinates, `= sin(x2a(j))` for spherical polar coordinates

Note that `x1a(i) < x1b(i)`. The exact relationship between the two grids is:

```
x1b(i) = x1a(i) + 0.5 * dx1a(i)
```

Similarly for the 2- and 3-directions. Every grid variable has a corresponding variable representing the quantity at the next time step and half way to the next time step. These are denoted by appending an "`n`" or an "`h`" respectively to the variable name. For example, `x1bn` and `x1bh` contain the values of `x1b` at the next time step and half time step respectively. Note that the three variables `x1b`, `x1bn`, and `x1bh` will be identical if the grid velocities are set to zero (a stationary grid). In addition, every grid variable has a corresponding inverse variable, denoted by appending an "`i`" to the variable name. Thus, `dx1ai=1/dx1a`, `x2bhi=1/x2bh`, *etc.* Evidently, there are numerous grid variables. However, only the a-grid variables `x1a`, `x2a`, and `x3a` are written to the restart dump. All others are re-computed when a job be resumed.

A3.2 FIELD VARIABLES (3-D ARRAYS)

There is very little internal scaling of variables in ZEUS-3D that the user must consider. Density, energy, and velocity all may be scaled according to the needs of the user simply by setting the initial conditions as appropriate. For example, the user may wish to set the density and the sound speed at infinity to unity. This, along with some canonical length scale will set the scale for the calculation. The only scaling implicit to ZEUS-3D

is the permeability of free space ($4\pi \times 10^{-7}$ in mks, 4π in cgs) is set to 1. Thus, the total pressure (thermal plus magnetic) is given by $p_{\rm tot} = p_{\rm thermal} + B^2/2$. Having set the scale of the hydrodynamical variables, the user should set the magnetic fields with this additional scaling in mind.

Variable	Location	Description
`d (i,j,k)`	zone center	density
`v1(i,j,k)`	1-face	velocity in the 1-direction (grid units)
`v2(i,j,k)`	2-face	velocity in the 2-direction (grid units)
`v3(i,j,k)`	3-face	velocity in the 3-direction (grid units)
`e (i,j,k)`	zone center	internal energy density ($\propto$ pressure)
`gp(i,j,k)`	zone-center	gravitational potential
`b1(i,j,k)`	1-face	magnetic field in the 1-direction ($\mu_0 = 1$)
`b2(i,j,k)`	2-face	magnetic field in the 2-direction ($\mu_0 = 1$)
`b3(i,j,k)`	3-face	magnetic field in the 3-direction ($\mu_0 = 1$)

If the EDITOR macro `ISO` is defined, the energy variable, `e`, is not declared. The gravitational potential variable, `gp`, and the magnetic field variables, `b1`, `b2`, `b3`, are declared only if the EDITOR macros `GRAV` and `MHD` are defined respectively.

A3.3 BOUNDARY VARIABLES (2-D ARRAYS)

First inner-i boundary

Variable	Location	Description
`niib (j,k)`		indicates boundary type for all variables (see discussion on namelist `iib` in Appendix 2.)
`diib1(j,k)`	zone-center at `i=is-1`	density
`v1iib1(j,k)`	1-face at `i=is`	1-velocity (normal to the boundary)
`v2iib1(j,k)`	2-face at `i=is-1`	2-velocity (tangential to the boundary)
`v3iib1(j,k)`	3-face at `i=is-1`	3-velocity (tangential to the boundary)
`eiib1(j,k)`	zone-center at `i=is-1`	internal energy density ($\propto$ pressure)
`gpiib(j,k)`	zone-center at `i=is-1`	gravitational potential
`b1iib1(j,k)`	1-face at `i=is`	1-magnetic field (normal to the boundary)
`b2iib1(j,k)`	2-face at `i=is-1`	2-magnetic field (tangential to the boundary)
`b3iib1(j,k)`	3-face at `i=is-1`	3-magnetic field (tangential to the boundary)

Second inner-i boundary

Variable	Location	Description
`diib2 (j,k)`	zone-center at `i=is-2`	density
`v1iib2(j,k)`	1-face at `i=is-1`	1-velocity (normal to the boundary)
`v2iib2(j,k)`	2-face at `i=is-2`	2-velocity (tangential to the boundary)
`v3iib2(j,k)`	3-face at `i=is-2`	3-velocity (tangential to the boundary)

`eiib2 (j,k)`	zone-center at `i=is-2`	internal energy density ($\propto$ pressure)
`b1iib2(j,k)`	1-face at `i=is-1`	1-magnetic field (normal to the boundary)
`b2iib2(j,k)`	2-face at `i=is-2`	2-magnetic field (tangential to the boundary)
`b3iib2(j,k)`	3-face at `i=is-2`	3-magnetic field (tangential to the boundary)

Analogous boundary variables exist at the outer-i boundary (`oib`), inner-j boundary (`ijb`), outer-j boundary (`ojb`), inner-k boundary (`ikb`), and outer-k boundary (`okb`). Note that the i-boundary variables use indices `(j,k)` and are declared so long as the EDITOR macro `ISYM` is *not* defined. Similarly, the j-boundary variables use indices `(k,i)` and are declared so long as `JSYM` is *not* defined. Finally, the k-boundary variables use indices `(i,j)` and are declared so long as `KSYM` is *not* defined. In addition, all energy boundary variables (`eiib1`, *etc.*) are *not* declared if `ISO` is defined and the gravity and magnetic boundary variables (`gpiib`, *etc.*; `b1iib1`, *etc.*) are declared *only if* `GRAV` and `MHD` are defined respectively. Note that the boundary variables are used only for regions of the boundary specified as "flow-in" (`niib(j,k)=3`). For all other boundary types (discussed in Appendix 2), the boundary values of the flow variables are determined from the values in the neighboring active zones.

A3.4 SCRATCH VARIABLES

As discussed in Subsection 6.1 (point 4), there are a multitude of scratch arrays available which can be used to minimize the additional memory required by the user's subroutines. These should be used wherever possible, especially for 3-D arrays. There are 26 1-D arrays dimensioned (`ijkn`) and named `wa1d` through `wz1d`. There are eight 2-D arrays dimensioned (`ijkn,ijkn`) and named `wa2d` through `wh2d`. Finally, there are six 3-D arrays dimensioned (`in,jn,kn`) and named `wa` through `wf`. For 2-D problems in which one of the EDITOR definitions `ISYM`, `JSYM`, `KSYM` is set, the 2-D arrays are dimensioned `(jn,kn)`, `(kn,in)`, `(in,jn)` respectively, and are equivalenced to `wa` through `wf`. The scratch arrays `wg2d` and `wh2d` are not declared when one of the symmetry flags is set.

A3.5 SUNDRY VARIABLES (AN ABBREVIATED LIST)

Variable	Description
`ioin`	logical unit attached to input deck
`iolog`	logical unit attached to message log file
`iotty`	logical unit attached to terminal (TTY or CRT)
`iodmp`	logical unit attached to restart dumps
`ioplt1`	logical unit attached to 1-D NCAR graphics dumps
`ioplt2`	logical unit attached to 2-D NCAR graphics dumps
`iopix`	logical unit attached to 2-D pixel dumps
`iovox`	logical unit attached to 3-D voxel dumps
`iousr`	logical unit attached to user dumps
`iotsl`	logical unit attached to time slice ascii dump
`iotslp`	logical unit attached to time slice plot dump

`iodis`	logical unit attached to display dump
`iorad`	logical unit attached to RADIO dump
`nhy`	number of cycles (time steps) completed in simulation
`nwarn`	running total of warnings issued
`time`	problem time elapsed in simulation
`dt`	increment of problem time that solution is being advanced

In addition, all of the namelist variables (except from namelist `pgen`) are declared in `rouse`.

A3.6 PARAMETERS (OTHER THAN ARRAY DIMENSIONS)

Parameter	Description
`niov`	maximum number of variables plotted/dumped
`nios`	maximum number of slices for each variable plotted/dumped
`ntsl`	maximum number of time slices to be collected for plots
`pi`	3.14159...
`tiny`	1.0×10^{-99}: smallest greater-than-zero number available on machine
`huge`	$1.0 \times 10^{+99}$: largest number available on machine

The parameter `nios` is used by the following I/O formats: 1-D NCAR plots, 2-D NCAR plots, pixel dumps, and display dumps. The parameter `niov` is used by all these I/O formats, plus: voxel dumps, HDF dumps, and RADIO dumps. They are both currently set to 20 in the common deck `par`, and can be altered as needed.

INDEX

Page numbers in **bold face** indicate primary references.

Batch (see NQS)
boundaries 13, 32, 34–35, 41, **49–53**, 74–75
CDBX (UNICOS utility) 11
change decks
  `chgz32` 9, 15, 17, **18**, 35–38
  `chgzeus` 9, 10, **15**, **17–18**, 19, 36, 39
compiler messages (`see zeus32.l`)
compilers
  CF77 (Cray) 15, 18
  CFT77 (Cray) 11, 15, 18, 33
  FC (Convex) 11
Convex (C2 and C3) 7, 8, 11, 12, 17, 22
CONVEXOS 7, 10, 12, 19
Courant condition 5, 14
Cray-2 and Cray-YMP 8, 9, 11, 12, 19, 20, 21, 22, 33
DADI algorithm (see garvity)
directory created (see `zeus3.2`)
display dumps 12, 14, 16, **25**, 29, **61–62**, 76
e-mail address of the Laboratory for Computational Astrophysics (LCA) 3
EDITOR 10, 17, 18–19, **35–38**
  `*alias, *al` **10**, 37
  `*call, *ca` 30, 33, **37**, 72
  `*deck, *dk, *cdeck, *cd` **30**, **33**, 36, 38
  `*define, *def` **9–10**, 11, 37
  `*delete, *d` **15**, 17, **36**, 38
  `*if, *else, *el, *endif, *ei` **32**, 34, **37**
  `*insert, *i` 35, **36**, 38
  `*read, *r` 15, **17**, 18
  `*replace, *rp` **36**
  aliases **13–14**, 19, 34, 40
  choices for aliases **12–14**, **36**
  comments **9**, **11**
  definitions **11–12**, 22–26, 34, 75
  error messages 37, 38
  `inedit` (input deck) **15**, 18, 19, 20, **36**
  listing files **35–36**
  merging files 10, 17, 19, 35, **36**
  precompiling files 11, 12, 19, **37–38**
  setting definitions and aliases **9–10**

EDITOR macros (see `zeus32.mac`)
energy, units of 34
equation of state **54**
  isothermal (EDITOR definition `ISO`) 7, 9, 11, 32, 48, 50–53, 54, 74, 75
  total energy equation 7, 16, **48**–**49**
executable (see `xzeus32`)
gravity 7, 11, 12, 13, 41, **53**, 75
GRAV (EDITOR definition) 7, 9, 11, 50–53, 74, 75
GRAVITY (EDITOR alias) 10, 11, **13**, 40, 41
grid extension 10, **13**, **55**
grid generation 13, 14, **16**, **45**–**49**, 72–74
HDF dumps 7, 12, 14, 16, 23, **24**, 25, 29, **62**
HISTORIAN 18, 35
`implicit none` statement 33
interrupt messages (see ZEUS-3D)
`inzeus` (input deck) 9, **16**, **19**–**20**, 34, 39, 42
listing `zeus32` (see EDITOR)
loaders
  NCARGF77 15, 18
make (UNICOS utility) 9, 11, 16, **20**
`makezeus` (makefile) 15, 16, 18, **20**, 35, 38
merging `zeus32` (see EDITOR)
message log file (`zl`*nn*`id`) **26**, 35, 75
MHD
  EDITOR macro 9, 11, 32, 34, 48, 50–53
  equations 4–5
multitasking (microtasking) 7, **18**–**19**, 33, 46
namelist.lib 15, **17**, 19
namelists 19–20, 22–26, 34, **42**
  column reserved for key characters 19, 42
  comments 18, 19, 42
  comparison between EDITOR and UNICOS versions **19**, **42**
  error messages 19
  setting rank 2 arrays 19
namelist parameters in ZEUS-3D 16, **42**–**70**, 76
NCAR graphics dumps 12, 14, 16, **22**–**23**, 55–57, 75
NQS (Batch) **20**–**21**, 39
parameters 10, **17**–**18**, 33–34, 76
  setting values in `zeus32` 12, **15**, **17**, 18, 39
pixel dumps 12, 14, 18, **23**–**24**, 29, **57**–**59**, 75
precompiling `zeus32` (see EDITOR)
problem generators 14, **30**–**33**, 41, **65**–**70**
  namelist `pgen` 16, 31, **34**
RADIO dumps 6, 7, 10, 12, 14, 18, **25**–**26**, 27, **62**–**65**, 76
restart dumps 7, 14, 15, **21**, 28–29, 41, **43**–**45**, 73, 75

restarting a run 14, 22, 41, **43–45**, 46–47, 56, 58, 60, 62, 63, 69
`rouse` (ZEUS-3D declarations) 30, **33**, 35, 71
scratch arrays in ZEUS-3D (see worker arrays)
script file (UNICOS utility, see also `zeus32.s`) 8, 15–16, 20, 36, 39
  comments 16
source code (see `zeus32`)
time slice dump 12, 14, **24–25**, 29, 61, 76
UNICOS 6, 7, 10, 12, 19, 42
USERDUMP 10, 14, 16, **26**, 29, 35, **60–61**, 76
variables (see ZEUS-3D)
voxel dumps 10, 14, 16, **24**, 29, **59–60**, 75
worker arrays in ZEUS-3D **33–34**, **75**
`xzeus32` (ZEUS-3D executable) 9, 15
  creating 8, **20**, 39
  executing **21–22**, 41
ZEUS-3D
  community code 4
  contributors 4
  data dumps 12, 14, **22–27**, 28–29, 48–54, 64, 65
    naming conventions **22–27**
  declarations (see `rouse`)
  features (new to version 3.2) **6–7**
  files required for compilation 8, 15, **17**
  geometry 5, 7, 9, 11, 71, 72–73
  initialising variables (see also problem generator) 12, 14, **30–35**
  interrupt messages **28–29**
  limitations 5–6
  MHD equations solved 4–5
  making changes
    adding whole subroutines 13–14, 18, 26, **30–35**, 71
    changes to existing code 8, 9, 15, 17–18, 25, **35–38**, 72–73
    debugging changes (see also `zeus32.l`) **20**
  numerical attributes 5
  scaling
    grid 45–47
    field variables 73–74
  size 4, 6
  skeleton 40
  source code (see `zeus32`)
  staggered grid 5, **72**
  variables 34, **71–76**
    boundary variables **74–75**
    field variables 31–32, 34, **73–74**
    grid variables **72–73**
    scratch variables 34, **75**

sundry variables **75**–**76**
`zeus.nqs` (see NQS)
`zeus3.2` directory 8, 11, 15, **17**, 18, 20, 35, 38
`zeus32` (source code) 9, 12, 15, **17**, 30, 33, 35, 36, 38, **40**, 42, 71
`zeus32.f` (EDITOR precompile of `zeus32`) **38**
`zeus32.l` (compiler messages) 9, 16, **20**
`zeus32.m` (EDITOR merge of `zeus32`) **37**
`zeus32.mac` (EDITOR macros) 8, **9**–**14**, **17**, 18, 34, 35, 39
`zeus32.n` (EDITOR listing of `zeus32`) **36**, 37
`zeus32.s` 8, 9, 10, 11, 12, **15**–**20**, 39, 42